VSB-TECHNICAL UNIVERSITY OF OSTRAVA

FACULTY OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

P H D T H E S I S

Study branch : Computer Science

Named Entity Recognition and Text Compression

Author : Vu Nguyen Hong

Supervisor : Prof. RNDr. Václav Snášel

Acknowledgements

This thesis is the result of research carried out during my PhD program at VSB-Technical University of Ostrava, Czech Republic. It is my pleasure to thank all those who have helped me.

First, I would like to express my deep appreciation to my academic and thesis supervisor, Professor Václav Snášel, who has been vigorously supervising my studies, supporting my research, and has been constantly involved in guiding me towards my goal. This thesis would not have been possible without his academic and insightful advice. It is priceless to me to have him as my supervisor. I would like to thank him for the consecutive trips to Vietnam to guide me so that I was able to achieve my goal of completing my research and this thesis project. I will never forget everything he has done for me since I arrived in the Czech Republic.

I am really grateful to Dr. Hien Nguyen Thanh, my second thesis supervisor, for his guidance, feedback, and comments during my research. He has given me advice and guided me how to approach and explore new challenges and how to divide an overwhelming task into smaller, more manageable tasks that are more readily accomplished. This allowed me to take my first steps in the world of research. During my research, he has always encouraged me, pushed me, and shared with me his experience, knowledge, and anything that he thinks will be valuable to me. I know that he spent a lot of his time with me and I wish to express my gratitude to him.

I am thankful to Dr. Phan Dao, Director of the European Cooperation Center of Ton Duc Thang University, Ho Chi Minh City, Vietnam, for giving me the opportunity to take part in the Sandwich Program. He has advised me on what to do and how can I achieve my goals during my research. I will never forget everything he did for me the first time I went to the Czech Republic; he and his family were so kind to help me arrange my accommodations, develop my itinerary, and choose some places to visit during my trip.

I am also thankful to my companion, Mr. Hieu Duong Ngoc, for his advice, support, and encouragement.

I would like to thank all of my colleagues, my friends, and my classmates in the Sandwich Program.

Finally, I wish to express my heartfelt gratitude to my family for their love, encouragement, and support; especially my beloved *Phuong Pham*.

Abstract

In recent years, social networks have become very popular. It is easy for users to share their data using online social networks. Since data on social networks is idiomatic, irregular, brief, and includes acronyms and spelling errors, dealing with such data is more challenging than that of news or formal texts. With the huge volume of posts each day, effective extraction and processing of these data will bring great benefit to information extraction applications.

This thesis proposes a method to normalize Vietnamese informal text in social networks. This method has the ability to identify and normalize informal text based on the structure of Vietnamese words, Vietnamese syllable rules, and a trigram model. After normalization, the data will be processed by a named entity recognition (NER) model to identify and classify the named entities in these data. In our NER model, we use six different types of features to recognize named entities categorized in three predefined classes: Person (PER), Location (LOC), and Organization (ORG).

When viewing social network data, we found that the size of these data are very large and increase daily. This raises the challenge of how to decrease this size. Due to the size of the data to be normalized, we use a trigram dictionary that is quite big, therefore we also need to decrease its size. To deal with this challenge, in this thesis, we propose three methods to compress text files, especially in Vietnamese text. The first method is a syllable-based method relying on the structure of Vietnamese morphosyllables, consonants, syllables and vowels. The second method is trigram-based Vietnamese text compression based on a trigram dictionary. The last method is based on an n-gram slide window, in which we use five dictionaries for unigrams, bigrams, trigrams, four-grams and five-grams. This method achieves a promising compression ratio of around 90% and can be used for any size of text file.

Keywords: text normalization, named entity recognition, text compression.

Contents

Chapter 1

Introduction

Contents

1.1 Motivation

In recent decades, along with the development of computer science and technology, the internet was also developed. Today, the internet has become one of the most popular channels for storing and transferring human information. The invention of the World Wide Web (the "Web") and its rapid development created a convenient opportunity for the distributing and sharing of information over internet. It led to the explosion of information in terms of quantity, quality, and subject. Two decades ago, the capacity of information was usually measured in MB or GB. However, in recent years, along with the appearance of big data theory, the common measurement units are now GB, TB, and PB. Almost all information on the web has been presented in natural language under the format of the HTML language. This language lacks the capability to express the semantics of concepts and objects presented on the web. Therefore, the majority of current information on the web is only suitable for humans to read and understand. From the objectives of effective mining of information resources from web, several applications to extract documents automatically were developed, such as information extraction systems, information retrieval systems, machine translations, text summarization, and question answering systems, etc. for computers to understand the semantics of sections of a text, instead of trying to understand the entire semantics of text. Some approaches have been proposed so that we can understand main entities and concepts appearing in the text based on source knowledge of entities and concepts in the real world.

Named entity recognition (NER) is a subtask of information extraction and one of the important parts of Natural Language Processing (NLP). NER is the task of detecting named entities in documents and categorizing them to predefined classes. Common classes of NER systems are person (PER), location (LOC), organization (ORG), date (DATE), time (TIME), currency (CUR), etc. For example, let us consider the following sentence:

On April 13, 2016, Mr. Hien Nguyen Thanh attended a meeting with CSC corporation in Ton Duc Thang University

In this sentence, an NER system would recognize and classify four named entities as follows:

- *April 13, 2016* is a date
- *Hien Nguyen Thanh* is a person
- *CSC corporation* is an organization
- *Ton Duc Thang University* is an organization

After the named entity has been recognized, it can be used for different important tasks. For example, it can be used for named entity linking and machine translation, such as we have on an iPhone device app, which takes a photo of a dish name on a menu, recognizes this name as an entity of dish names and foods, maps it to a knowledge source about entities and concepts in the real world, such as Wikipedia[1]. The app then translates it to the user's language. Normally, from the recognized entities, other mining systems can be built to mine new classes of knowledge and get a better result than the raw text.

Many approaches have been proposed for NER from the first conference, the 6th Message Understanding Conference (MUC-6) in 1995. In this conference, the NER task was first introduced and was subsequently discussed at the next conference, the Conference on Computational Natural Language Learning (CoNLL) in 2002 and 2003. Most of them focused on English, Spanish, Dutch, German, and Chinese according to the data set from conferences and the popularity of these languages. In the domain of Vietnamese, several approaches have been proposed and were presented detail in 2.3.3. However, none apply to Vietnamese informal text.

In this dissertation, we propose a method to fill that gap. We started research on NER for Vietnamese informal texts in the middle of 2014, and specifically focused on Vietnamese tweets on Twitter. When studying Vietnamese tweets, we found that they contained many spelling errors, typing errors, which created a significant challenge for NER. To overcome this challenge, we studied the Vietnamese language, normalization techniques, and proposed a method to normalize Vietnamese tweets in [Nguyen 2015b]. After we normalized these tweets, we proposed a method to recognize name entities in [Nguyen 2015a].

According to statistics from 2011, the number of tweets was up to 140 million per day[2]. With such a huge number of tweets being posted every day, it raised a storage challenge. Regarding this challenge, in [Nguyen 2015b], we used a trigram language model, which size is rather large compared with other methods. Therefore, we want to save its storage too. When researching this challenge, we found that

[1] http://www.wikipedia.org
[2] https://blog.twitter.com/2011/numbers

there have not been any text compression methods proposed for Vietnamese. After studying several methods for text compression, we proposed the first approach for Vietnamese text compression based on syllable and structure of Vietnamese in [Nguyen 2016a]. In this approach, the compression ratio is converged to around 73%. It is still low when compared with other methods and especially, this method has a high compression ratio for a small text file. From this disadvantage, we continued researching and proposed a method based on trigram in [Nguyen 2016b], and the compression ratio of this method shows significant improvement when compared with the previous method. Its compression ratio is around 82%, and it is still not the best. In the next approach, we propose a method based on the n-gram slide window and achieve an encouraging compression ratio of around 90%. This is higher than the two previous methods and with other methods. One important significance of this method, however, is that it can apply to any size of text file.

1.2 Thesis objective and scope

The objectives of this thesis are briefly summarized as follows.

1. To suggest a method to compress Vietnamese text based on Vietnamese morphosyllable structure, such as syllables, consonants, vowels, and marks; Vietnamese dictionary of syllables with their marks; Vietnamese dictionary of consonants, vowels, etc.

2. To propose a method to compress Vietnamese text based on the trigram language model.

3. To propose a method to compress text based on n-gram sliding windows.

4. To propose a method to detect Vietnamese errors in informal text, especially focused on Vietnamese tweets on Twitter, and to normalize them based on a dictionary of Vietnamese morphosyllables, Vietnamese morphosyllable structures, and Vietnamese syllable rules in the combination with language model.

5. To propose an NER model to recognize named entities in Vietnamese informal text, especially focused on Vietnamese tweets on Twitter.

1.3 Thesis organization

The rest of the dissertation is structured as follows.

Chapter 2 describes the background and related work. In this chapter, we present the structure of Vietnamese words, and the typing methods to compose Vietnamese words. We also present the theory of text compression, text compression techniques, some NER techniques, and some related work involving to our research.

In Chapter 3, we present some techniques for Vietnamese text compression. They are a syllable-based method, a trigram-based method, and finally an n-gram based method.

In Chapter 4, we offer a model and technique to normalize Vietnamese error informal text based on Vietnamese morphosyllable structure, syllable rules, and a trigram dictionary. We also propose a method to improve the Dice coefficient in [Dice 1945].

In Chapter 5, we propose a model and technique to recognize named entities in Vietnamese informal text focusing on Vietnamese tweets on Twitter. In our model, we recognize three categories of named entities including person, organization, and location.

Finally, Chapter 6 is our conclusions and future work.

Chapter 2

Background and related work

Contents

2.1 Vietnamese language processing resources

2.1.1 Structure of Vietnamese word

Currently, there are several viewpoints on what is a Vietnamese word. However, in order to meet the goals of automatic error detection, normalization and classification, we followed the viewpoint in [Thao 2007], i.e., "A Vietnamese word is composed of special linguistic units called Vietnamese morphosyllable." Normally, it has from one to four morphosyllables. A morphosyllable may be a morpheme, a word, or neither of them [Tran 2007]. For example, in a sample Vietnamese sentence "Sinh viên Trường Đại học Tôn Đức Thắng rất thông minh," there are eleven morphosyllables, i.e., "Sinh," "viên," "Trường," "Đại," "học," "Tôn," "Đức," "Thắng," "rất," "thông," and "minh."

According to the syllable dictionary of Hoang Phe [Phe 2011], a morphosyllable has two basic parts, i.e., consonant and syllable with mark, or one part, i.e., syllable with mark. We describe more detail for these parts in followings:

- Consonant: The Vietnamese has 27 consonants, i.e., "b," "ch," "c," "d," "đ," "gi," "gh," "g," "h," "kh," "k," "l," "m," "ngh," "ng," "nh," "n," "ph," "q," "r,"

"s," "th," "tr," "t," "v," "x," and "p." In those consonants, there are eight tail consonants, i.e., "c," "ch," "n," "nh," "ng," "m," "p," and "t."

- Syllable: A syllable may be a vowel, a combination of vowels, or a combination of vowels and tail consonants. According to the syllable dictionary of Hoang Phe, the Vietnamese language has 158 syllables, and the vowels in these syllables do not occur consecutively more than once, except for the syllables "ooc" and "oong."
- Vowel: The Vietnamese has 12 vowels, i.e., "a," "ă," "â," "e," "ê," "i," "o," "ô," "ơ," "u," "ư," and "y."
- Mark: The Vietnamese has six marks, i.e., not marked (none) ("a"), acute accent ("á"), grave accent ("à"), hook above ("ả"), tilde accent ("ã"), and dot bellow ("ạ"), which are marked above or below a certain vowel of each syllable.

2.1.2 Typing methods

There are two popular typing methods used to compose Vietnamese, i.e., Telex typing and VNI typing. Each method combines letters to form Vietnamese morphosyllables. Vietnamese characters have some extra vowels that do not exist in Latin characters, i.e., â, ă, ê, ô, ơ, ư, one more consonant, đ; Vietnamese has six types of marks as mention above. The combination of vowels and marks forms the Vietnamese language its own identity.

- When using Telex typing, we have the combination of characters to form Vietnamese vowels, such as aa for â, aw for ă, ee for ê, oo for ô, ow for ơ, and uw for ư. Also we have one consonant, dd for đ. For forming marks, we have s for acute accent, f for grave accent, r for hook above, x for tilde accent, and j for dot below.
- Similar to Telex typing, we have the combination of characters in VNI typing, such as a6 for â, a8 for ă, e6 for ê, o6 for ô, o7 for ơ, u7 for ư, and d9 for đ. To form marks, we have: 1 for acute accent, 2 for grave accent, 3 for hook accent, 4 for tilde accent, and 5 for heavy accent.

For example, to compose morphosyllable *trường*, the normal Telex typing combines sequence of these letters *truwowngf*, sometimes we can change the order of these letter such as *truowngf, truwowfng, truwfowng*. The normal VNI typing combines sequence of these letters *tru7o7ng2*, sometimes we can change the order of these letter such as *truo7ng2, truo72ng, tru72o7ng*.

2.1.3 Standard morphosyllables dictionary

We synthesize a standard morphosyllables dictionary of Vietnamese by combining all consonants, syllables, and marks. This dictionary includes 7,353 morphosyllables in lowercase and the size is around 51 KB.

2.2 Text compression

2.2.1 Introduction to text compression

According to [Salomon 2010], data compression is the process of converting an input data stream (the source stream or the original raw data) into another data stream (the output, the bitstream, or the compressed stream) that has a smaller size. A stream can be a file, a buffer in memory, or individual bits sent on a communications channel. The main objectives of data compression are to reduce the size of input stream and increase the transfer rate as well as save storage space. Figure 2.1 shows the general model of data compression. In this model, the input data stream X will be encoded to compress stream Y, which has a smaller size than X. Data stream Z is the stream recovered from compressed stream Y.

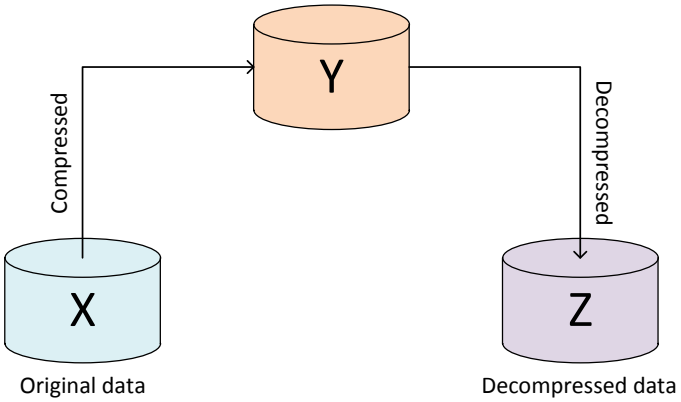

Figure 2.1: Data compression model

Data compression techniques are classified into two classes, i.e., lossless and lossy compression. Using the lossless compression technique, after encoding, a user can completely recover the original data stream from a compressed stream, whereas with lossy compression, a user cannot retrieve exactly the original data stream, as the retrieved file will have some bits lost. According to Figure 2.1, if the compression technique is lossless, the data stream Z is the same as input stream X; otherwise, if the compression technique is lossy, the data stream Z is different from input stream X. Based on the property of compression classification, lossy compression techniques archive a higher compression ratio than lossless. Normally, lossy compression techniques achieve a compression ratio from 100:1 to 200:1, while lossless compression techniques achieve a compression ratio from 2:1 to 8:1. The compression ratio of both lossy and lossless compression techniques depend on the type of data stream being compressed.

Based on the purposes of the compression task, the user will have a suitable choice for compression technique. If they can accept that the decompression data stream will be different from the input data stream, they can choose the lossy

compression technique to save on storage. In the case that the decompression data stream and its input need to be the same, the user must use lossless compression technique.

Text compression is a field of data compression, which uses the lossless compression technique to convert an input file (sometimes called the source file) to another form of data file (normally called the compressed file). It cannot use the lossy compression technique because it needs to recover the exact original file from the compressed file. If it uses lossy, the meaning of the input file and the file recovered from the compressed file will be different. Several techniques have been proposed for text compression in recent years. Most of them are based on the same principle of removing or reducing the redundancy from the original input text file. The redundancy can appear at character, syllable, or word level. This principle proposed a mechanism for text compression by assigning short codes to common parts, i.e., characters, syllables, words, or sentences, and long codes to rare parts.

2.2.2 Text compression techniques

There are several techniques developed for text compression. These techniques can be further classified into four major types, i.e., substitution, statistical, dictionary, and context-based. The substitution text compression techniques replace a certain longer repeating of characters with a shorter one. A technique that is representative of these techniques is run length encoding [Robinson 1967]. The statistical techniques usually calculate the probability of characters to generate the shortest average code length, such as Shannon-Fano coding [Shannon 1948, Fano 1949], Huffman coding [Huffman 1952] and and arithmetic coding [Witten 1987, Howard 1994]. The next type is dictionary techniques, such as Lempel-Ziv-Welch (LZW), which involves substitution of a sub-string of text by indices or pointer code relating to a position in dictionary of the substring [Ziv 1977, Ziv 1978, Welch 1984]. The last type is context-based techniques, which involve the use of minimal prior assumptions about the statistics of the text. Instead, they use the context of the text being encoded and the past history of the text to provide more efficient compression. Representatives for this type are prediction by partial matching (PPM) [Cleary 1984] and Burrow–Wheeler transform (BWT) [Burrows 1994]. Every method has its own strengths, weaknesses, and is applied to a specific field, and none of the above methods has been able to achieve the best results in terms of compression ratio.

2.2.3 Related work

In recent years, most text compression techniques have been based on dictionary, word level, character level, syllable-based, or BWT. [Al-Bahadili 2008] proposed a method to convert the characters in the source file to a binary code, where the most common characters in the file have the shortest binary codes, and the least common have the longest. The binary codes are generated based on the estimated probability of the character within the file and are compressed using 8-bit

character word length. [Kalajdzic 2015] proposed a technique to compress short text messages based on two phases. In the first phase, it converts the input text consisting of letters, numbers, spaces, and punctuation marks commonly used in English writing to a format which can be compressed in the second phase. In the second phase, it proposes a transformation which reduces the size of the message by a fixed fraction of its original size. In [Platos 2008a], the authors proposed a word-based compression variant based on the LZ77 algorithm, and proposed and implemented various ways of sliding windows and various possibilities of output encoding. In a comparison with other word-based methods, their proposed method is the best. In these studies, they do not consider the structure of words or morphemes in the text. In [Lansky 2005], Lansky and his colleagues were the first to propose a method for syllable-based text compression techniques. In their paper, they focused on specification of syllables, methods for decomposition of words into syllables, and using syllable-based compression in combination of the principles of LZW and Huffman coding. Recently, [Akman 2011] presented a new lossless text compression technique which utilizes syllable-based morphology of multi-syllabic languages. The proposed method is designed to partition words into its syllables and then to produce their shorter bit representations for compression. The number of bits in coding syllables depends on the number of entries in the dictionary file. In [Platoš 2008b], the authors first proposed a method for small text file compression based on the Burrow–Wheeler transformation. This method combines the Burrow–Wheeler transform with the Boolean minimization at the same time.

Unfortunately, most of the proposed methods are applied to language other than Vietnamese. For Vietnamese text, according to our best knowledge, no text compression has been proposed.

2.3 Named entity recognition

2.3.1 Introduction

Named entity recognition task was first introduced at MUC-6 in 1995. In this conference, NER consists of three subtasks. The first task is ENAMEX, which detects and classifies proper names and acronyms which are categorized via the three types as follows:

- ORGANIZATION: named of corporate, governmental, or other organizational entity such as “Ton Duc Thang University”, “CSC Corporation”, “Gia Dinh hospital”.
- PERSON: named person or family such as “Phuong Pham Thi Minh”, “Han Nguyen Vu Gia”.
- LOCATION: name of politically or geographically defined location (cities, provinces, countries, international regions, bodies of water, mountains, etc.) such as “Ho Chi Minh City”, “New York”.

The second task is TIMEX, which detects and classifies of temporal expressions which are categorized in two types as follows:

- DATE: complete or partial date expression such as "April 2016", "April 24, 2016".
- TIME: complete or partial expression of time of day such as "six p.m.", "1h30 a.m.".

The last task is NUMEX, which detects and classifies of numeric expressions, monetary expressions and percentages which are categorized in two types as follows:

- MONEY: monetary expression such as "9,000 VND", "10,000 USD".
- PERCENT: percentage such as "20%", "ten percent".

The example below, cited from [Grishman 1996], shows a sample sentence with named entity annotations. In this example, "Dooner" was annotated as a person, "Ammirati & Puris" was annotated as an organization, and "$400 million" was annotated as money, etc.

Mr. <ENAMEX TYPE= "PERSON">Dooner</ENAMEX> met with <ENAMEX TYPE= "PERSON"> Martin Puris </ENAMEX>, president and chief executive officer of <ENAMEX TYPE= "ORGANIZATION"> Ammirati & Puris </ENAMEX>, about <ENAMEX TYPE= "ORGANIZATION"> McCann </ENAMEX>'s acquiring the agency with billings of <NUMEX TYPE ="MONEY"> $400 million </NUMEX>, but nothing has materialized.

In CoNLL 2002[1] and CoNLL 2003[2], the shared task concerned language independent NER. In these two conferences, they concentrate on four types of named entities: persons, locations, organizations, and names of miscellaneous entities that do not belong to the previous three groups. The participants of the shared task have been offered training and test data for two European languages: Spanish and Dutch in CoNLL 2002 [Tjong Kim Sang 2002] and for two other European languages: English and German in CoNLL 2003 [Tjong Kim Sang 2003].

In recent years, because of the development of social networks, several approaches have been proposed for NER in social networks. One approach was shown at the Shared Tasks of the 2015 Workshop on Noisy User-generated Text: Twitter Lexical Normalization and Named Entity Recognition. This workshop has two parts, the first is Twitter lexical normalization and the second is NER over Twitter [Baldwin 2015]. In this shared task, the participants concentrate on ten types of named entities such as company, facility, geo-loc, movie, music artist, person, sportsteam, product, tvshow, and other. The training and test data include 1,795 annotated tweets for training and 599 as a development set.

[1] http://www.cnts.ua.ac.be/conll2002/ner/
[2] http://www.cnts.ua.ac.be/conll2003/ner/

2.3.2 NER techniques

Currently, there are several techniques developed for NER. Based on the properties of the proposed techniques, they can be categorized in four types, including i) knowledge-based or rule-based technique, ii) statistical technique, iii) machine learning technique, and iv) hybrid technique.

Knowledge-based systems normally are based on rules [Humphreys 1998, Mikheev 1998, Cunningham 1999, Nguyen 2007a]. Rules in these systems were created by humans and sometimes are considered as hand-crafted deterministic rule-based systems. These rules can be built from dictionaries, regular expressions, or context-free grammar. Dictionaries of named entities in the NER system are often called Gazetteers. It is difficult to build a full gazetteer. Therefore, these methods that use gazetteers usually combine with other methods to have a more complex system. The system with rules created from context-free grammar often depends on a particular domain or a specific language; it is not a portable system. Therefore, when we want to apply it to a new field or a new language, we must modify rules. These tasks require a lot of time and money. It requires that the authors have expert knowledge in this field and this language.

Machine learning systems can be categorized in three main classes including supervised learning, semi-supervised learning, and unsupervised learning. Several techniques have been proposed for supervised learning, including Hidden Markov Models [Bikel 1997], Support Vector Machines [Asahara 2003], Maximum Entropy [Borthwick 1998], and Conditional Random Fields [McCallum 2003]. These techniques were used to create rules automatically from training set as presented in [Tjong Kim Sang 2002, Tjong Kim Sang 2003]. Normally, these methods are more flexible and robust than rule-based methods. When we need to apply a method of machine learning to a new field, the machine learning method needed to be trained on new training set to be suitable with the new field. Moreover, there are some rules that are missing when building a rule set, which can be determined and generated by machine learning methods. Although flexible and robust, the supervised learning method has a limit that it requires a data set, in which the named entities are annotated, large enough, and high quality. Therefore, it requires more effort to build a training set. To overcome these limitations, semi-supervised learning techniques have been proposed. Semi-supervised learning [Riloff 1999] requires a training set in which the named entities were annotated, having a small size. These named entities are then used to find patterns or contexts around them. New named entities are then found using these patterns and contexts. This process is then iterated and the new named entities are always used as patterns for the next step. Therefore, this method is often called bootstrapping. Besides the learning techniques mentioned above, unsupervised learning techniques [Collins 1999, Etzioni 2005] also have been proposed. This technique does not require any training set to recognize named entities. They are typically based on clustering. The clustering methods can use contexts, patterns etc., and rely on a large corporation.

The hybrid methods [Mikheev 1999, Mikheev 1998] typically combine between

two or three methods from above to achieve a better result.

2.3.3 Related work

a. NER

NER has been studied extensively on formal texts, such as news and authorized web content. Several approaches have been proposed using different learning models, such as Condition Random Fields (CRF), Maximum Entropy Model (MEM), Hidden Markov Model (HMM), and Support Vector Machines (SVM). In particular, [Mayfield 2003] used SVM to estimate lattice transition probabilities for NER. [McCallum 2003] applied a feature induction method for CRF to recognize named entities. A combination of a CRF model and latent semantics to recognize named entities was proposed in [Konkol 2015]. A method using soft-constrained inference for NER was proposed in [Fersini 2014]. In [Curran 2003] and [Zhou 2002], the authors proposed a maximum entropy tagger and an HMM-based chunk tagger to recognize named entities. Unfortunately, those methods gave poor performance on tweets, as pointed out in [Liu 2011].

b. Vietnamese NER

In the domain of Vietnamese texts, various approaches have been proposed using various learning models, such as SVM [Tran 2007], classifier voting [Thao 2007] and CRF [Le 2011, Tu 2005]. Some other authors have proposed other methods for NER, such as a rule-based method [Nguyen 2010, Nguyen 2007b], labeled propagation [Le 2013a], the use of a bootstrapping algorithm and a rule-based model [Trung 2014], and combined linguistically-motivated and ontological features [Nguyen 2012b]. [Pham 2015] proposed an online learning algorithm, i.e., MIRA [Crammer 2003] in combination with CRF and bootstrapping. [Sam 2011] used the idea of Liao and Veeramachaneni in [Liao 2009] based on CRF and expanded it by combining proper name co-references and named ambiguity heuristics with a powerful sequential learning model. [Le 2013b] proposed a feature selection approach for named entity recognition using a genetic algorithm. To calculate the accuracy of the recognition of the named entity, this paper used KNN and CRF. [Nguyen 2012a] proposed a systematic approach to avoid the conflict between rules when a new rule was added to the set of rules for NER. [Le 2015] proposed some strategies to reduce the running time of genetic algorithms used in a feature selection task for NER. These strategies included reducing the size of the population during the evolution process of the genetic algorithm, reducing the fitness computation time of individuals in the genetic algorithm by using progressive sampling for finding the (near) optimal sample size of the training data, and parallelization of individual fitness computation in each generation.

Table 2.1: Result of several previous works in Vietnamese NER

System	Entity Types	Precision	Recall	F1
[Le 2011]	PER	84% %	82.56%	83.39%
[Nguyen 2010]	PER, ORG, LOC, NA, FA, RE	92%	76%	83%
[Nguyen 2007b]	PER, ORG, LOC	86.05%	81.11%	83.51%
[Sam 2011]	PER, ORG, LOC	93.13%	88.15%	79.35%
[Thao 2007]	PER, ORG, LOC, CUR, NUM, PERC, TIME	86.44%	85.86%	89.12%
[Tran 2007]	PER, ORG, LOC, CUR, NUM, PERC, TIME	89.05%	86.49%	87.75%
[Tu 2005]	PER, ORG, LOC, CUR, NUM, PERC, TIME, MISC	83.69%	87.41%	85.51%

However, there have been no approaches that focused on NER in Vietnamese tweets or (short) informal Vietnamese texts.

In order to better collocate our results with other existing Vietnamese NER systems that used other techniques, we report the performances of other Vietnamese NER systems in Table 2.1. The meanings of the abbreviations in Table 2.1 are listed here: PER: Person, ORG: Organization, LOC: Location, CUR: Currency, NUM: Number, PERC: Percent, TIME: Time, NA: Nationality, FA: Facility, RE: Region, MISC: Miscellaneous.

c. NER in tweets

Regarding microblog texts written in English and other languages, several approaches have been proposed for NER. Among them, [Ritter 2011] proposed a NER system for tweets, called T-NER, which employed a CRF model for training and Labled-LDA. [Ramage 2009] proposed an external knowledge base, i.e., Freebase[2] for NER. A hybrid approach to NER on tweets was presented in [Liu 2011] in which a KNN-based classifier and a CRF model were used. A combination of heuristics and MEM was proposed in [Jung 2012]. In [Tran 2015], a semi-supervised learning approach that combined the CRF model with a classifier based on the co-occurrence coefficient of the feature words surrounding the proper noun was proposed for NER on Twitter. [Li 2015a] proposed non-standard word (NSW) detection and decided a word is out of vocabulary (OOV) based on the dictionary, and then applied the normalization system of [Li 2014] to normalize OOV words. The results from NSW

[2]http://www.freebase.com

detection was used for NER based on the pipeline strategy or the joint decoding fashion method. In [Liu 2013b], a named entity was recognized using three steps, i.e., 1) each tweet is pre-labeled using a sequential labeler based on the linear Conditional Random Fields (CRFs) model; 2) tweets are clustered to put those that have similar content into the same group; and 3) each cluster refines the labels of each tweet using an enhanced CRF model that incorporates the cluster-level information. [Liu 2012b] proposed jointly conducting NER and Named Entity Normalization (NEN) for multiple tweets using a factor graph, which leverages redundancy in tweets to make up for the dearth of information in a single tweet and allows these two tasks to inform each other. [Liu 2013a] proposed a novel method for NER consisting of three core elements, i.e., normalization of tweets, combination of a KNN classifier with a linear CRF model, and a semi-supervised learning framework. [Nguyen 2012c] presented a method for incorporating global features in NER using re-ranking techniques that used two kinds of features, i.e., flat and structured features and a combination of CRF and SVM. In [Zirikly 2015], a CRF model without being focused on Gazetteers was used for NER for Arabic social media.

Recently, [Baldwin 2015] presented the results of Shared Tasks of the 2015 Workshop on Noisy User-generated Text: Twitter Lexical Normalization and Named Entity Recognition. According to this paper, most of researchers used CRF. However, several researchers in this workshop described new methods, such as [Godin 2015]], which used absolutely no hand-engineered features and relied entirely on embedded words and a feed-forward, neural-network (FFNN) architecture; [Cherry 2015] developed a semi-Markov MIRA trained tagger; [Yamada 2015] used entity-linking-based features, and other researchers used CRFs.

Since some of the specific features of Vietnamese were presented in [Tran 2007], one cannot apply those methods directly to Vietnamese tweets.

Chapter 3

Vietnamese text compression

Contents

3.1 Introduction

In 2012, every day 2.5 EB of data were created and in 2015, every minute we have nearly 1,750 TB of data being transferred over the internet, according to a report from IBM[1] and the forecast of Cisco[2], respectively. Reducing the size of data is an effective solution to increasing the data's transfer rate and saving storage space.

[1] http://www-01.ibm.com/software/data/bigdata/what-is-big-data.html

[2] http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.html

As mentioned in 2.2, there are several methods proposed for text compression. However, all of them are focused on languages other than Vietnamese. Therefore, in this chapter, we propose some methods for Vietnamese text compression. The first method is based on syllables, marks, and syllable dictionaries. This method builds several dictionaries to store all syllables of the Vietnamese language. Each dictionary has been built based on the structure of syllables and marks. Because the total number of Vietnamese syllables is 158, we use 8 bits to represent all syllables of each dictionary above. In the compression phase, each morphosyllable will be split into a consonant and a syllable, and then they will be processed and analyzed to select the appropriate dictionary to get the corresponding index. This index will be replaced for the consonant and syllable in the text file. Depending on the structure of the morphosyllable, we encode it using two or three bytes. To decompress the encoded morphosyllable, we read the first byte and analyze it to decide how many bytes need to be read next. Finally, we will decode it to get the original morphosyllable. The second method is based on the trigram dictionary of the Vietnamese language. This method first splits the input sequence to trigrams then encodes them based on the trigrams dictionary. With each trigram, it uses four bytes to encode. The last method proposes a method for Vietnamese text compression based on n-gram. In this method, it first splits Vietnamese text to n-grams, then encodes them based on the n-grams dictionaries. In the encoding phase, we use a sliding window that has a size from bi-gram to five-gram to slide the input sequence to have the best encoding stream. With each n-gram, it uses two to four bytes to encode according to its corresponding dictionary.

This chapter presents the first attempt at Vietnamese text compression. The rest of this chapter is organized as follows. In section 3.2, 3.3, and 3.4, we give a detailed description of the syllable-based method, trigram-based method, and n-gram-based method, respectively. Finally, we present our summaries in Section 3.5.

3.2 A syllable-based method for Vietnamese text compression

In this section, we present a syllable-based method for Vietnamese (SBV) text compression. This method has two main phases. The first phase is SBV text compression and the second phase is SBV text decompression. Figure 3.1 describes our method model. In our model, we use dictionaries and morphosyllable rules for both two phases. We will describe more details about it in following subsections.

3.2.1 Dictionary

In our method, we use several dictionaries for both compression and decompression phases. These dictionaries have been built based on the combination between the syllables and marks. Because the total number of syllables and consonants is less

than 256, for each dictionary we use the maximum 8 bits to represent. Table 3.1 describes the structure and number of entries of each dictionary.

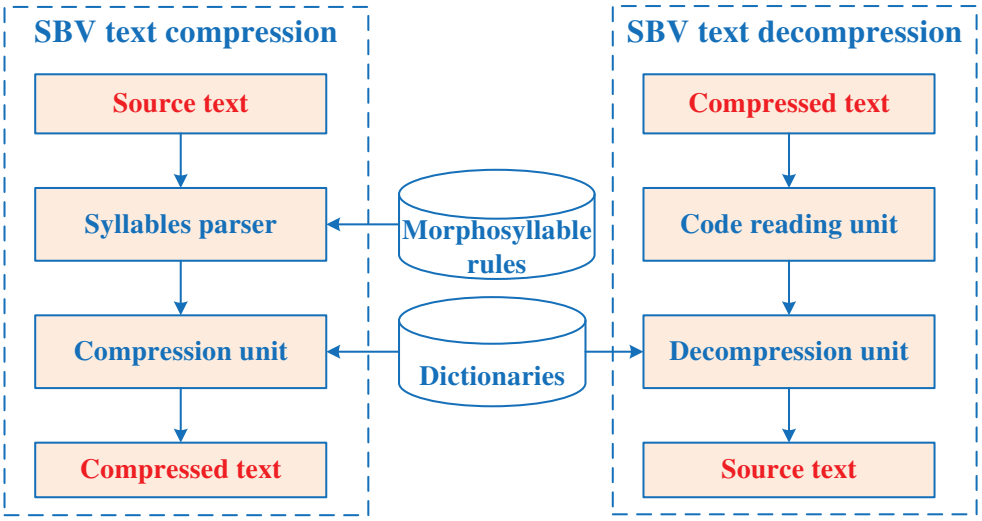

Figure 3.1: a syllable-based Vietnamese text compression model

Table 3.1: Dictionaries structure

No.	Index	Dictionary	Number of Entries	Number of bits
1	0	Acute accent	178	8
2	1	Dot below	176	8
3	2	None	158	8
4	5	Tilde accent	114	7
5	6	Grave accent	112	7
6	7	Hook above	111	7
7		Consonants	27	5
8		Special characters	42	6

There may appear to be cases where there are multiple capital letters for all characters of morphosyllable or capital letters of the first character of a syllable, non-standard word, e.g., email-id, or web link. We will handle this case by case and discuss more details in the following subsections.

3.2.2 Morphosyllable rules

a. Identifying consonant and syllable

According to section 2.1.1, Vietnamese morphosyllable has two basic parts: consonant and syllable with mark. In this section, we propose a rule to split the consonant and the syllable with mark. According section 2.1.1, we will build a table of consonants, including 27 consonants, as in Table 3.2. To split the consonant and syllable, we search the value of each consonant in the consonants dictionary Table 3.2 with morphosyllable. If it is found, then we can split the morphosyllable to the consonant and syllable with mark based on the length of the consonant in the consonants dictionary.

Table 3.2: Consonants dictionary

index	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**
value	**ngh**	**ng**	**gi**	**gh**	**kh**	**nh**	**ph**	**th**	**tr**	**k**	**g**	**h**	**l**	**m**

index	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**24**	**26**
value	**n**	**q**	**r**	**s**	**t**	**v**	**x**	**p**	**b**	**ch**	**c**	**d**	**đ**

For example, we have the morphosyllable “nguyễn”. To split the consonant and syllable with mark of this morphosyllable, we search each value of consonants dictionary refer from Table 3.2 from index zero with morphosyllable. In this example, it is found that the value “ng” occurred in this morphosyllable, and the index of value “ng” in the consonants dictionary is 1. The length of “ng” is two; therefore, we can split this morphosyllable from the letter, of which the position in the morphosyllable is two. Notice that the position begins from zero. In this case, the morphosyllable, “nguyễn”, can be split into the consonant, “ng”, and the syllable with mark, “uyễn”.

b. Identifying mark

According to section 2.1.1, Vietnamese has six types of marks. Because we split a morphosyllable into a consonant and syllable with mark, we must know what the mark of this syllable is to map it with the mark to find the correct syllable with mark in the dictionary. To do that, we built a table of 12 vowels and six marks. Refer to Table 3.3.

Table 3.3: Vietnamese vowels and their marks

mark	index	0	1	2	3	4	5	6	7	8	9	10	11
acute accent	0	á	ấ	ắ	é	ế	í	ó	ớ	ố	ú	ứ	ý
dot below	1	ạ	ậ	ặ	ẹ	ệ	ị	ọ	ợ	ộ	ụ	ự	ỵ
none	2	a	â	ă	e	ê	i	o	ơ	ô	u	ư	y
tilde accent	5	ã	ẫ	ẵ	ẽ	ễ	ĩ	õ	ỡ	ỗ	ũ	ữ	ỹ
grave accent	6	à	ầ	ằ	è	ề	ì	ò	ờ	ồ	ù	ừ	ỳ
hook above	7	ả	ẩ	ẳ	ẻ	ể	ỉ	ỏ	ở	ổ	ủ	ử	ỷ

Every syllable just has one mark. So, to identify the mark of the syllable, we can search each value of Vietnamese vowels and their marks in Table 3.3, starting from column index zero and acute accent row with vowel. The return result is the value of the index column corresponding to the value of the index column of Table 3.1, for example, with syllable "uyễn". The system will search values of Vietnamese vowels and their marks in Table 3.3. According to this case, the system found that the value "ễ" appears in syllable "uyễn", the value of the index corresponding to this vowel is five. This means that the mark of this syllable is the tilde accent.

c. Identifying capital letter

To identity the capital letter in consonant and syllable, we use a capitals dictionary to store all the capital letters of single consonants and vowels with their marks, similarly to vowels in Table 3.3, but in capital form. To identify capital letters of consonants, step by step we search each single consonant of this consonant in the capitals dictionary. If it is found in the capitals dictionary, we record the position of this single consonant in the consonant. We use the same method to identify the capital letters of the syllable.

3.2.3 SBV text compression

According to Figure 3.1, the SBV text compression phase has two main parts, the first part is the syllables parser and the second is the compression unit. In the following subsections, we will focus on the details.

a. Syllables parser

The syllables parser has been used to separate morphosyllables in the input sequences, splitting morphosyllables into consonants and syllables. It is also used to classify each syllable to the corresponding dictionary and detect the capitalization of characters in consonants and syllables.

We separate morphosyllables based on the spaces character. In this stage, we also classify the morphosyllable to a standard morphosyllable or non-standard morphosyllable based on the Vietnamese dictionary of standard morphosyllables from section 2.1.3. A morphosyllable will be classified as non-standard if it does not appear in this dictionary. Before classifying a morphosyllable as a standard or non-standard morphosyllable, we must convert all characters of morphosyllable to lowercase.

Parse for standard morphosyllables
With each standard morphosyllable received from the separating morphosyllable task, the syllables parser splits it into consonant and syllable, assigns the capital property for them, and classifies them to the corresponding dictionary. This task can be described as follows:

1. Splitting morphosyllables to consonant and syllable is based on the structure of Vietnamese morphosyllable and morphosyllable rules.

2. Adding a position attribute for uppercase characters of consonant and syllable:

    - Because the number of characters in consonants is less than or equal to three, we use three bits to represent the position of capital letters of consonants and call it consonant property. If the character of consonants is a capital character, the value of its bit is 1, or 0 for a lowercase character. For example, we have some consonants and corresponding consonant property of them: **NGH - 111, ngH - 001, nGH - 011**.
    - In the case of syllables, because the number characters of the syllables are less than four characters, we use four bits to represent the position of capital characters of syllables and call it syllable property. Similarity to a consonant, if the character of the syllable is a capital character, the value of its bit is 1, or 0 for a lowercase character. For example, we have some syllables and the corresponding syllable property for them: **ƯỜNG - 1111, ưỜNG - 0111, ƯờNG - 1011**.

3. Classifying the syllable into the corresponding dictionary is based on identifying mark rules.

Parse for non-standard morphosyllables
With non-standard morphosyllables, we classify them to one of two classes as follows:

1. Special characters: if one of their characters appears in the special character dictionary.

2. Other: the character does not appear in the special character dictionary.

b. Compression unit

The compression unit uses the results from the syllables parser, detecting consonants and syllables in dictionaries to find their corresponding code. Based on the structure of the syllable, consonant, and property of the character, if it has a capital letter or not. We will use two or three bytes to encode a morphosyllable. The compression task can be summarized as follows:

Two bytes encoding
A morphosyllable will be encoded by two bytes in these following cases:

1. A morphosyllable does not have capital letter in it and mark of the syllable is different from a tilde.
2. A special character occurs in the special character dictionary.

The two bytes encoding has a structure like below:
$\mathbf{0\ B_1^6\ B_1^5\ B_1^4\ B_1^3\ B_1^2\ B_1^1\ B_1^0\ B_2^7\ B_2^6\ B_2^5\ B_2^4\ B_2^3\ B_2^2\ B_2^1\ B_2^0}$
Where:

- **0**: two bytes encoding.
- $\mathbf{B_1^6\ B_1^5\ B_1^4\ B_1^3\ B_1^2}$:Encode for the index of the consonant in consonants dictionary. We use five bits because the number entry of the consonants dictionary is 27 (refer to Table 3.2).
- $\mathbf{B_1^1\ B_1^0\ B_2^7}$: Encode for the index of the syllable mark in the dictionary from Table 3.1. We use three bits because we have six marks, but we move the tilde to three bytes encoding, so, we only have five marks. According to Table 3.1, the number of entries of acute accent, heavy accent, and none in the dictionary is greater than **128**. So, we just use $\mathbf{B_1^1\ B_1^0}$ to encode for the index of these dictionaries in Table 3.1, we move $\mathbf{B_2^7}$ to the next part to encode for the position of the syllable in dictionary. When the mark is a grave accent or a hook accent, we use all three bits $\mathbf{B_1^1\ B_1^0\ B_2^7}$ which value is **110** and **111** corresponding to the index of these marks in the dictionary in Table 3.1, respectively.
- $\mathbf{B_2^7\ B_2^6\ B_2^5\ B_2^4\ B_2^3\ B_2^2\ B_2^1\ B_2^0}$: Encode for the position of the syllable in the dictionary. As presented above, $\mathbf{B_2^7}$ is used in the case the mark is an acute accent, dot below, or none.
- In the case of a special character, we will set all bits of $\mathbf{B_1^6\ B_1^5\ B_1^4\ B_1^3\ B_1^2\ B_1^1\ B_1^0}$ to **1**, $\mathbf{B_2^7\ B_2^6\ B_2^5\ B_2^4\ B_2^3\ B_2^2\ B_2^1\ B_2^0}$ will present for the position of special character in its corresponding dictionary.

Three bytes encoding
A morphosyllable will be encoded with three bytes when it is a standard morphosyllable and it has at least one capital letter or the mark of the syllable is a tilde.

The three bytes encoding has a structure like below:
1 B_1^6 B_1^5 B_1^4 B_1^3 B_1^2 B_1^1 B_1^0 B_2^7 B_2^6 B_2^5 B_2^4 B_2^3 B_2^2 B_2^1 B_2^0 B_3^7 B_3^6 B_3^5 B_3^4 B_3^3 B_3^2 B_3^1 B_3^0
Where:

- **B_1^6 B_1^5 B_1^4 B_1^3 B_1^2**: Encode for the position of the consonant in the dictionary.
- **B_1^1 B_1^0 B_2^7**: Encode for the capital characters of the consonant. We use three bits because the number characters of a consonant are less than or equal to three.
- **B_2^6 B_2^5 B_2^4**: Encode for mark.
- **B_3^7 B_3^6 B_3^5 B_3^4 B_3^3 B_3^2 B_3^1 B_3^0**: Encode for the position of the syllable in the dictionary.
- **B_2^3 B_2^2 B_2^1 B_2^0**: Encode for the capital characters of the syllable. We use four bits because the number characters of a syllable are less than or equal to four.

Other cases: unknown number of bytes
In the case of a non-standard morphosyllable and it is not a special character, we will encode it with the entire non-standard morphosyllable. To distinguish from the two cases above, we add two more bytes. The first byte has a value of 255 to designate it is a special case of non-standard word. The value of the second byte is the length of a non-standard morphosyllable.

3.2.4 SBV text decompression

SBV text decompression is the inversion of the SBV text compression phase. The SBV text decompression phase is undergone in two steps and can be summarized as follows:

- **Code reading unit**: This unit is used to read output sequence from the compression result as their input sequence separates it byte by byte.
- **Decompression unit:**
  This unit is used to decode morphosyllables one by one. To do that, it reads one byte from the output sequence of the code reading unit. It analyzes this byte and decides how many bytes will be read more based on the first bit of this byte. There are two cases for this situation.
  - If the first bit of the first byte is 0, the decompression unit will read **one** byte more from the output sequence of the code reading unit and decodes it. This task is the inversion of two bytes encoding.
  - If the first bit of the first byte is 1:

* If all remaining bits of the first byte is not equal to 1, the decompression unit reads **two** bytes more from the output sequence of the code reading unit and decodes it. This task is the inversion of three bytes encoding.
* If all bits of the first byte are 1: the decompression unit reads one byte more to decide how many bytes will be read more based on the value of this byte. This task is the inversion of the special case of non-morphosyllable encoding.

After finishing the decoding for one morphosyllable, it will read the next byte, repeat the decompression task to decode another morphosyllable until it has read all the way to the last byte.

3.2.5 Compression ratio

Compression ratio is used to measure the efficiency of compression method, the higher the compression ratio the higher quality of compression method. Normally, we use unicode encoding to present Vietnamese text. Every character in Unicode is stored in two bytes. The compression ratio can be calculated by equation 3.1.

$$CR = \left(1 - \frac{\text{compressed\_file\_size}}{\text{original\_file\_size}}\right) \times 100 \tag{3.1}$$

Where:

- original_file_size = (total number of characters in original file) x 2
- compressed_file_size = total number of bytes in compressed file

3.2.6 Example

Assuming that we use the following dictionaries for this example, e.g., the consonants dictionary shown in Table 3.2, the syllable dictionaries shown in Table 3.4:

Table 3.4: syllable dictionaries

	acute accent			grave accent	heavy accent		none			
index	0	1	2	0	0	1	0	1	2	3
value	ất	ắng	ức	ường	ại	ọc	iên	inh	ôn	ông

a. SBV text compression

Assuming that we want to compress the input sequence *Sinh viên Trường Đại học TÔN ĐỨC THẮNG rất thông minh* (students of TON DUC THANG University are very intelligent), this input sequence has 11 morphosyllables.

Syllables parser

The syllables parser first separates all morphosyllables in this input sequence to 11 morphosyllables like that: *Sinh*, *viên*, *Trường*, *Đại*, *học*, *TÔN*, *ĐỨC*, *THẮNG*, *rất*, *thông*, *minh*. Then, based on the dictionary of standard Vietnamese morphosyllables in section 2.1.3, it classifies these morphosyllables to standard morphosyllables. Because all morphosyllables are standard morphosyllables, the syllables parser will parse them for the standard morphosyllables case. The syllable parser encounters

Table 3.5: output of syllables parser

No	Morphosyllable	Consonant	PCLC	Syllable	PCLS	IOM
1	**Sinh**	**S**	100	**inh**	0000	2
2	**viên**	**v**	000	**iên**	0000	2
3	**Trường**	**Tr**	100	**ường**	0000	6
4	**Đại**	**Đ**	100	**ại**	0000	1
5	**học**	**h**	000	**ọc**	0000	1
6	**TÔN**	**T**	100	**ÔN**	1100	2
7	**ĐỨC**	**Đ**	100	**ỨC**	1100	0
8	**THẮNG**	**TH**	110	**ẮNG**	1110	0
9	**rất**	**r**	000	**ất**	0000	0
10	**thông**	**th**	000	**ông**	0000	2
11	**minh**	**m**	000	**inh**	0000	2

the first morphosyllable of eleven morphosyllables, *Sinh*, and splits this morphosyllable into consonant and syllable based on the identifying consonant and syllable rules. After this step, it receives consonant *S* and syllable *inh*. To identify the capital letter attribute of the consonant and syllable, it uses identifying capital letter rules. According to these rules, *S* is a capital letter; therefore, the value of three bits presents that the capital letter positions of the consonant will be *100*. When applying these rules to syllable *inh*, it realizes that this syllable does not have any capital letters in it; therefore, the value of four bits present for the capital letter positions of the syllable will be *0000*. To identify the dictionary for syllables with marks, it uses the identifying mark rules. In this case, it receives result two after using the identifying mark rules. Therefore, the index value for the dictionary of syllables with marks is two. It repeats this step until it reaches the last morphosyllable. Table 3.5 presents the results of the syllable parser with the input sequence mentioned above. In this table, we have some abbreviations and meanings as follows: IOM: index of mark of syllable in Table 3.3, PCLC: position of capital letters

of the consonant, PCLS: position of capital letters of the syllable. In Table 3.5, the morphosyllable *THẮNG* has a consonant of *TH* and its syllable is *ẮNG*. Both *T* and *H* are capital letters; therefore, the consonant capital letter property of this consonant is *110*. Similarly to syllable *ẮNG*, it has three capital letters; therefore, the value of the syllable capital letter property is *1110*. In the syllable *ẮNG*, there is an acute accent, so the corresponding index of this mark in the dictionary in Table 3.3 is zero.

Compression unit

The compression unit uses results from the syllables parser as its input. With the first morphosyllable, *Sinh*, it has a capital letter. Therefore, the compression unit uses three-byte encoding to encode for this morphosyllable. According to three-bytes encoding, the morphosyllable *Sinh* will be compressed as follows:

- Encode for the index of the consonant in the consonants dictionary. The compression unit searches the consonant *s* in the consonant dictionary from Table 3.2. The index of this consonant is 17. It is presented in binary by the sequence *10001*.
- Encode for the position of the capital letters of the consonant. According to the result from the syllables parser, the encoding sequence is *100*
- Encode for the index of the syllable mark from Table 3.1. According to the result from the syllables parser, the index value is 2. Therefore, the encoding sequence in binary is *010*.
- Encode for the index of the syllable in the syllables dictionary. According to Table 3.4, the index of this syllable is one, so the encoding sequence for the index of the syllable in binary is *00000001*.
- Encode for the position of capital letters in the syllable. According to the result from the syllables parser, the encoding sequence in binary is *0000*.
- Finally, the result of the encoding sequence of the morphosyllable *Sinh* is *110001100010000000010000*. The first bit of this sequence is 1, meaning that it is three-byte encoding.

Next, the syllables parser compresses for the next morphosyllable, *viên*, which does not have a capital letter; therefore, it uses two-byte encoding to encode for this morphosyllable. According to two-bytes encoding, the morphosyllable *viên* will be compressed as follows:

- Encode for the index of the consonant in the consonant dictionary. The compression unit searches the consonant *v* in the consonant dictionary from Table 3.2. The index of this consonant is 19. It is presented in binary by the sequence *10011*.

Table 3.6: Compression result

No	MPS	ICCD	PCLC	IOM	ISIM	PCLS	Encoded sequence
1	Sinh	10001	100	010	00000001	0000	110001100010000000010000
2	viên	10011		10	00000000		0100111000000000
3	Trường	01000	100	110	00000000	0000	101000100110000000000000
4	Đại	11010	100	001	00000000	0000	011010100001000000000000
5	học	01011		01	00000001		0010110100000001
6	TÔN	10010	100	010	00000010	1100	110010100010000000101100
7	ĐỨC	11010	100	000	00000010	1100	111010100000000000101100
8	THẮNG	00111	110	000	00000001	1110	100111110000000000011110
9	rất	10000		00	00000000		0100000000000000
10	thông	00111		10	00000011		0001111000000011
11	minh	01101		10	00000001		0011011000000001

- Encode for the index of the syllable's mark in Table 3.1. According to the result from the syllables parser, the index value is two. Therefore, the encoding sequence in binary is *10*. Notice that in this case we just use two bits to present.
- Encode for the index of the syllable in the marks dictionary. According to Table 3.4, the index of this syllable is zero, so the encoding sequence for the index of this syllable in binary is *00000000*.
- Finally, the result of the encoding sequence of the morphosyllable *viên* is *0100111000000000*. The first bit of this sequence is 0, meaning that it is two-byte encoding.

These steps will be repeated until the last morphosyllable is reached. Table 3.6 shows the compression result of all morphosyllables of the input sequence. In this table, we have some abbreviations and meanings as follows: MPS: morphosyllable, ICCD: index of consonant in the consonant dictionary, PCLC: position of capital letters of consonants, IOM: index of mark of syllables in Table 3.3, PCLS: position of capital letters of the syllable, ISIM: index of the syllable in the marks dictionary.

The final encoder output sequence is the result of concatenation of all encoding output sequences from steps one to eleven in Table 3.6. The final encoder output sequence is:
110001100010000000010000|0100111000000000|101000100110000000000000|01101010000100000000000|0010110100000001|110010100010000000101100|1110101000

00000000101100/10011111000000000011110/0100000000000000/0001111000000011/0011011000000001

b. SBV text decompression

In this section, we will take the output sequence from SBV text compression and decode it using the SBV text decompression. The output sequence from the previous part is:
11000110001000000010000/0100111000000000/1010001001100000000000000/0110101000010000000000000/0010110100000001/1100101000100000000101100/1110101000000000000101100/10011111000000000011110/0100000000000000/0001111000000011/0011011000000001

First, the code reading unit reads the output sequence from the SBV text compression as its input. It separates this input byte by byte. Then, the decompression unit decodes for each morphosyllable, one by one according to the output of code reading unit.

To decode for the first morphosyllable, the decompression unit reads the first byte of the input sequence, the content of this first byte being *11000110*, the first bit of this byte is *1*, and all remaining bits of this byte are different from *1*. Therefore, it reads the next two bytes and decodes this morphosyllable based on these three bytes, so the value of these three bytes is *110001100010000000010000*. The decode of this morphosyllable can be described as follows:

- Decode for the consonant. First, the decompression unit calculates the index of the consonant to identify the consonant in the consonant dictionary, Table 3.2. To do that, it reads five bits from the second position to six of the three-byte input, giving this five bit a binary of *10001*, and a value of 17. Therefore, the index of the consonant in the consonant dictionary is 17. The value of this index in the consonant dictionary is *s*. Next, it finds the positions of the capital letter in the consonant. It reads the next three bits of the three bytes input, and the three bits is *100*. The decompression unit realizes that the first bit of this three bits is *1*, the next two bits is *0*. Therefore, the first letter of the consonant is capitalized, and the consonant will become *S*.

- Decode for syllable. First, it calculates the index of the mark of the syllable to identify the syllable dictionary, Table 3.4. To do that, it reads next three bits of three bytes input, where the three bits is *010*, and the value of the three bits is two. Therefore, the index of the syllable's mark is two, corresponding to none dictionary in Table 3.4. Then, it calculates the index of the syllable in the syllables dictionary based on the next eight bits, where these eight bits are *00000001* and their value is one. The value of this index in none dictionary in Table 3.4 is *inh*. Next, it finds the positions of the capital letter in the syllable. The decompression unit reads the next four bits of three bytes input, and those four bits are *0000*. It realizes that all of four bits are *0*, meaning that they do not have any capital letters in the syllable.

- The decoded morphosyllable is the concatenation of the encoder consonant and syllable. The value of the morphosyllable is *Sinh*. It is the same as the first morphosyllable of input sequence.

The decompression unit will repeat the decoding steps above to decode for all remaining morphosyllables and will get the same sequence as the input sequence.

c. Compression ratio

According to the input sequence, total number of characters in this sequence is 53, so the size of this sequence is 53 x 2 bytes and the size of the compressed sequence according to Table 3.6 is 28 bytes. According to equation 3.1, we have:

$$\mathbf{CR} = \left(\mathbf{1} - \frac{28}{53\times 2}\right) \times \mathbf{100} = \mathbf{73.59}\%$$

From the compression ratio above, we found that the size of the compressed result decreases to 73.59%.

3.2.7 Experiments

We conduct experimens to evaluate our method. We present our experiment results in Table 3.7 and Table 3.8. We also compress these input files using WinRAR version 5.21[3] (the software combines LZSS [Storer 1982] and Prediction by Partial Matching [Cleary 1984]) and WinZIP version 19.5[4] (the software combines LZ77 [Ziv 1978] and Huffman coding) to have a comparison. Table 3.7 shows the results of our method in 10 cases with different sizes and content of input files. The size of text files that we use in Table 3.7 is smaller than 15 KB. According to the results of Table 3.7 and Figure 3.2, our compression ratio is better than WinRAR and WinZIP. In these cases, our compression ratio is around 73%. In Table 3.7 and 3.8, Figure 3.2 and 3.3, we have some abbreviations and meanings as follows: OFS: original file size, CFS: compressed file size, CR: compression ratio.

[3] http://www.rarlab.com/download.htm
[4] http://www.winzip.com/win/en/index.htm

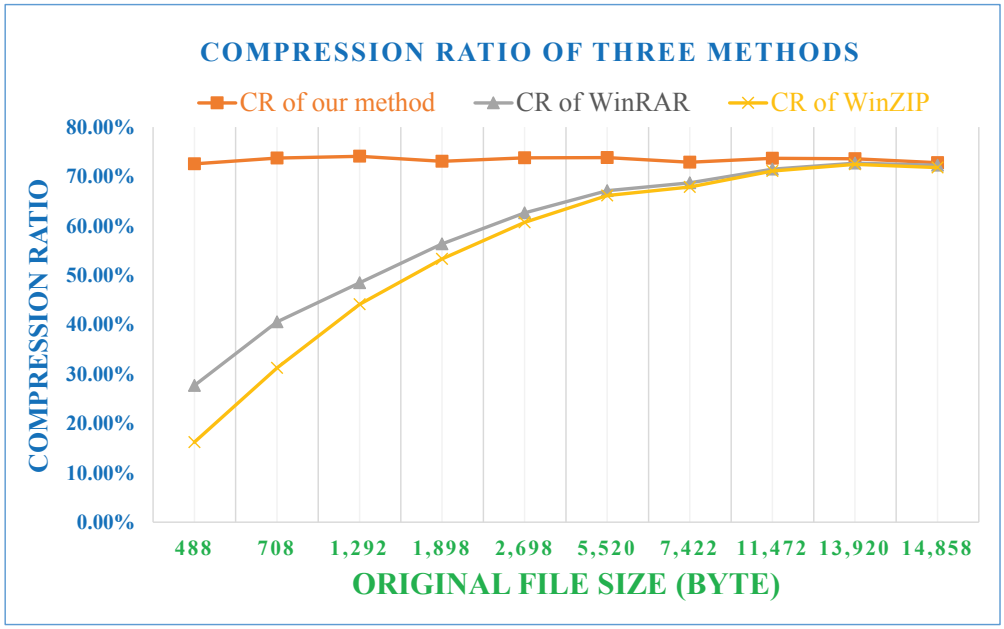

Figure 3.2: Compression ratios for file size smaller than 15 KB

Table 3.7: Experimental results for file size smaller than 15 KB

No	OFS (Byte)	CFSM (Byte)	CRM	CFSR (Byte)	CRR	CFSZ (Byte)	CRZ
1	**488**	**134**	**72.55%**	**353**	**27.67%**	**409**	**16.19%**
2	**708**	**186**	**73.73%**	**421**	**40.54%**	**484**	**31.25%**
3	**1,292**	**335**	**74.08%**	**666**	**48.45%**	**722**	**44.12%**
4	**1,898**	**511**	**73.08%**	**829**	**56.33%**	**886**	**53.32%**
5	**2,698**	**708**	**73.76%**	**1,008**	**62.64%**	**1,061**	**60.68%**
6	**5,520**	**1,445**	**73.83%**	**1,816**	**67.10%**	**1,870**	**66.13%**
7	**7,422**	**2,012**	**72.90%**	**2,323**	**68.71%**	**2,385**	**67.87%**
8	**11,472**	**3,020**	**73.68%**	**3,273**	**71.47%**	**3,316**	**71.10%**
9	**13,920**	**3,676**	**73.60%**	**3,802**	**72.69%**	**3,836**	**72.45%**
10	**14,858**	**4,045**	**72.78%**	**4,124**	**72.25%**	**4,189**	**71.81%**

In Table 3.8, we show that the result of our method in 10 cases with the size of text files is lager than 15 KB. According to the results, our compression ratio is lower than WinRAR and WinZIP.

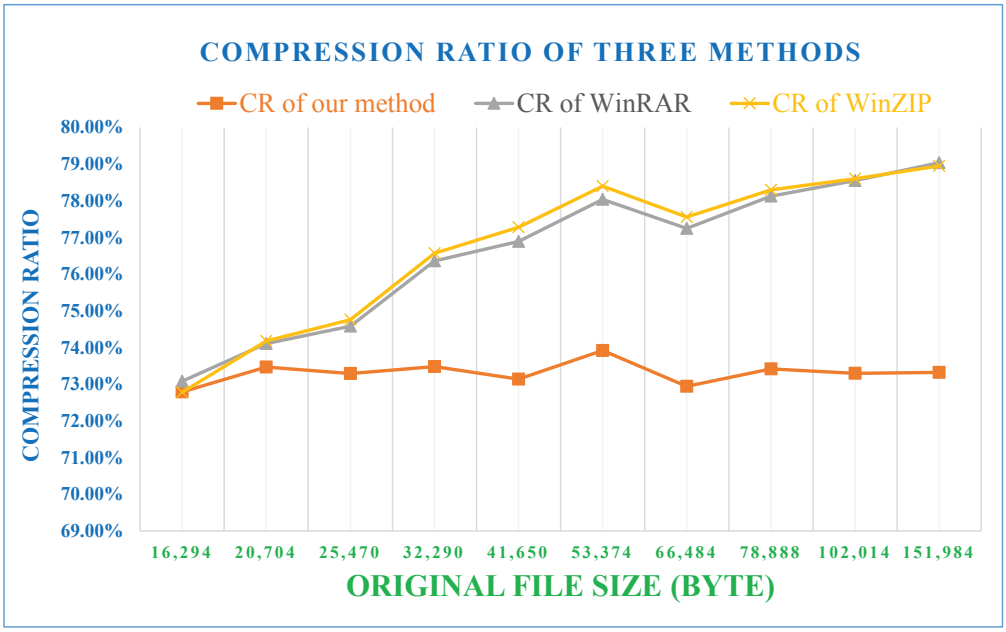

Figure 3.3: Compression ratios for file size larger than 15 KB

Table 3.8: Experimental results for file size larger than 15 KB

No	OFS (Byte)	CFSM (Byte)	CRM	CFSR (Byte)	CRR	CFSZ (Byte)	CRZ
11	**16,294**	**4,434**	**72.79%**	**4,404**	**73.08%**	**4,436**	**72.78%**
12	**20,704**	**5,493**	**73.47%**	**5,361**	**74.11%**	**5,346**	**74.18%**
13	**25,470**	**6,804**	**73.29%**	**6,475**	**74.58%**	**6,429**	**74.76%**
14	**32,290**	**8,565**	**73.48%**	**7,636**	**76.36%**	**7,567**	**76.57%**
15	**41,650**	**11,189**	**73.14%**	**9,629**	**76.89%**	**9,467**	**77.28%**
16	**53,374**	**13,921**	**73.92%**	**11,721**	**78.04%**	**11,530**	**78.40%**
17	**66,484**	**17,997**	**72.94%**	**15,128**	**77.25%**	**14,920**	**77.56%**
18	**78,888**	**20,969**	**73.42%**	**17,259**	**78.13%**	**17,120**	**78.30%**
19	**102,014**	**27,236**	**73.30%**	**21,879**	**78.55%**	**21,835**	**78.60%**
20	**151,984**	**40,552**	**73.32%**	**31,865**	**79.04%**	**32,002**	**78.95%**

According to all results of our experiments, our method achieves a higher compression ratio when the file size is smaller than 15 KB. Especially, when the file size is smaller than 2.5 KB, the compression ratio of our method is more efficient than

WinRAR and WinZIP. In these cases, in comparison with WinRAR and WinZIP, the compression ratio of our method is higher than 10%. Therefore, this method can apply efficiency to compress for Vietnamese short text such as SMS messages and text messages on social networks.

3.3 Trigram-based Vietnamese text compression

Although the compression ratio of the syllable-based method is very high, it converges to a ratio around 73%. Because of the structure of Vietnamese morphosyllables, it is very hard to improve this ratio if we still use this method. Therefore, in this section, we propose a new method for Vietnamese text compression called Trigram-based Vietnamese (TGV) text compression. Figure 3.4 describes our method model. In our model, we use a trigrams dictionary for both compression and decompression.

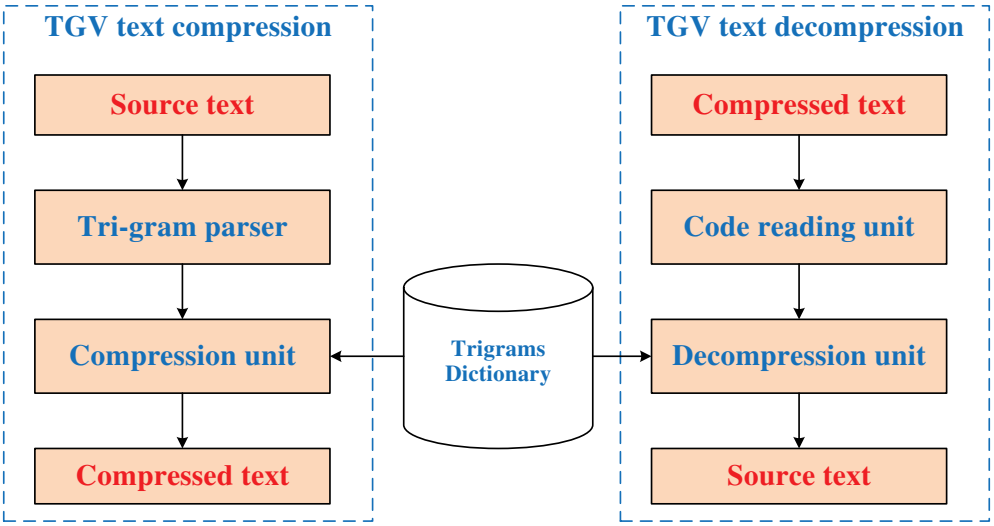

Figure 3.4: Trigram-based Vietnamese Text Compression

3.3.1 Dictionary

In our method, we build two trigrams dictionaries with different sizes to evaluate the effects of dictionary size with our method. Each dictionary has two columns, one contains trigrams and one contains the index for these trigrams. These dictionaries were built based on a text corpus collected from open access databases. The size of the text corpus for the dictionary one is around 800 MB and for dictionary two is around 2.5 GB. We use SRILM to generate the trigram data for these dictionaries. The trigrams data after using SRILM for dictionary one is around 761 MB with more than 40,514,000 trigrams and for dictionary two is around 1,586 MB with more than 84,000,000 trigrams. To reduce the search time in dictionaries, we arranged

them according to the alphabet. Table 3.9 describes the size and number of trigrams of each dictionary.

Table 3.9: Dictionaries

Dictionary	Number of tri-grams	Size (MB)
one	**40,514,822**	**761**
two	**84,003,322**	**1,586**

3.3.2 TGV text compression

According to Figure 3.4, the compression phase has two main parts, the first part is a trigrams parser and the second is a compression unit. In the following subsections, we will explain them more in detail.

a. Tri-grams parser

The trigrams parser is used to read the source text file, separate it into sentences based on newline and splits all sentence text into trigrams. In the case of the last trigram, maybe it just has a unigram or bigram. Therefore, we must assign an attribute to it to distinguish the trigram from the unigram and bigram. In Table 3.11, the value of this attribute is one.

b. Compression unit

The compression unit uses the results from the trigram parser, and detects each trigram in the dictionary to find the corresponding index for standard trigrams. If a trigram occurs in dictionary, we encode it using four bytes, otherwise we encode it with exactly the number of characters that it has. The compression task can be summarized as follows.

Encoding for tri-grams in the dictionary
Compression unit searches the trigram in the trigrams dictionary to get the index of trigrams if found, otherwise return 0. When a trigram occurs in the dictionary, we use four bytes to encode it. To distinguish with trigrams that do not occur in the dictionary and bigram and unigram, the compression unit sets the most significant bit of the first byte to zero. So, the four bytes encoding has the following structure:
$\mathbf{0\ B_0^6\ B_0^5\ B_0^4\ B_0^3\ B_0^2\ B_0^1\ B_0^0\ B_1^7\ B_1^6\ B_1^5\ B_1^4\ B_1^3\ B_1^2\ B_1^1\ B_1^0\ B_2^7\ B_2^6\ B_2^5\ B_2^4\ B_2^3\ B_2^2\ B_2^1\ B_2^0\ B_3^7\ B_3^6\ B_3^5\ B_3^4\ B_3^3\ B_3^2\ B_3^1\ B_3^0}$
Where:

- The most significant bit of the first byte is *0*: Encode for a trigram which occurs in the dictionary.

- $\mathbf{B_0^6\ B_0^5\ B_0^4\ B_0^3\ B_0^2\ B_0^1\ B_0^0\ B_1^7\ B_1^6\ B_1^5\ B_1^4\ B_1^3\ B_1^2\ B_1^1\ B_1^0\ B_2^7\ B_2^6\ B_2^5\ B_2^4\ B_2^3\ B_2^2\ B_2^1\ B_2^0\ B_3^7\ B_3^6\ B_3^5\ B_3^4\ B_3^3\ B_3^2\ B_3^1\ B_3^0}$: Encode for the index of the trigram in the dictionary.

Encoding for trigrams do not occur in the dictionary and for other cases When a trigram does not occur in the dictionary and for other cases, e.g., unigram and bigram, the compression unit encodes it using exactly number of characters that it has. In this case, it sets the most significant bit of the first byte to one. The next seven bits of this byte will present the number of bytes of this trigram and other cases in Unicode encoding because the Vietnamese language was presented in Unicode encoding. So, the encoding structure of this case was described as follows.
$\mathbf{1\ B_0^6\ B_0^5\ B_0^4\ B_0^3\ B_0^2\ B_0^1\ B_0^0\ B_i^7\ B_i^6\ B_i^5\ B_i^4\ B_i^3\ B_i^2\ B_i^1\ B_i^0}$
Where:

- The most significant bit of the first byte is *1*: Encode for a tri-gram which does not occur in the dictionary and for other cases.
- $\mathbf{B_0^6\ B_0^5\ B_0^4\ B_0^3\ B_0^2\ B_0^1\ B_0^0}$: Number of bytes of this trigram or other cases in Unicode encoding.
- $\mathbf{B_i^7\ B_i^6\ B_i^5\ B_i^4\ B_i^3\ B_i^2\ B_i^1\ B_i^0}$: Encoded bytes of trigram characters that do not occur in the dictionary and other cases. For our testing data, we use the Vietnamese language and normally it is presented by Unicode encoding. In the encoding stream, we use Unicode encoding. So, the value of i is the number of bytes that Unicode encoding uses to encode this trigram or other cases.
- We set all values of $\mathbf{B_0^6\ B_0^5\ B_0^4\ B_0^3\ B_0^2\ B_0^1\ B_0^0}$ to 1 to encode for a newline. Therefore, to encode for a newline we just use one byte.

3.3.3 TGV text decompression

TGV text compression is the inversion of TGV compression phase. The TGV text decompression process has undergone in two steps and can be summarized as follows.

- Code reading unit: this unit reads the encoding sequence from TGV text compression, and separates it byte by byte.
- Decompression unit: this unit uses the output sequence from the code reading unit as its input. It decodes one by one trigram or other cases, e.g., where unigram, bigram or trigram does not occur in the dictionary. The decompression unit decides to decode the trigram or other cases based on the first bit of the first byte of the input sequence at each step it decodes for new trigram or other cases. If the first bit is *0*, it decodes the trigram that occurred in the dictionary. Otherwise, it decodes for other cases. We describe the detail of the decompression unit as follows.

- The most significant bit of first byte is 0; it decodes for trigram that occurred in the dictionary. The decompression unit reads the next three bytes, calculates the index of this trigram, searches the trigram corresponding to this index. This task is the inversion of trigram occurred in dictionary encoding.
- The most significant bit of first byte is 1, it decodes for other cases.
  * If all remaining bits of the first byte are not equal to 1, then the decompression unit calculates the value of the remaining bits of first byte. This value is the number of bytes that it needs to read more from the input sequence. It decodes for these bytes based on Unicode decoding.
  * If all remaining bits of first byte are 1, then this is the encoded newline. The compression unit decodes a newline for it.

After finishing the decoding for one trigram or other cases, it reads the next byte, and repeats the decompression task to decode for other trigrams or other cases until it reads to the last byte.

3.3.4 Example

Assuming that we use the dictionary in Table 3.10 for this example. According to this dictionary, it has three standard trigrams.

Table 3.10: Trigrams dictionary

Index	Tri-gram
0	**Tôi là sinh**
1	**viên Trường Đại**
2	**học Tôn Đức**

a. TGV text compression

Assume that we want to compress this sequence *Tôi là sinh viên Trường Đại học Tôn Đức Thắng*. In the first stage, the trigrams parser splits all text of this input sequence to trigrams. This input sequence has three standard trigrams and one unigram. The trigrams parser results for this input are presented in Table 3.11.

Table 3.11: Trigram parser results

No.	Tri-gram	Attribute
1	**Tôi là sinh**	**0**
2	**viên Trường Đại**	**0**
3	**học Tôn Đức**	**0**
4	**Thắng**	**1**

After parsing for all trigrams, the output of this step is sent to the compression unit to encode. First, the compression unit encodes for the first trigram *Tôi là sinh*. It searches this trigram in the trigrams dictionary, Table 3.10. This trigram occurs in this dictionary and its index is zero. So, it sets the first bit of encoding sequence to *0*. The next 31 bits are the binary string of the trigram index in this dictionary, which in this case is zero. Therefore, the encoding sequence of this trigram is *00000000000000000000000000000000*. Then, it repeats this task to the last trigrams or other cases.

Table 3.12: TGV text compression results

No.	tri-grams or other cases	Encoded sequence
1	**Tôi là sinh**	**00000000000000000000000000000000**
2	**viên Trường Đại**	**00000000000000000000000000000001**
3	**học Tôn Đức**	**00000000000000000000000000000010**
4	**Thắng**	**10000111010101000110100011100001** **10111010101011110110111001100111**

The last trigram of the input sequence is a non-standard trigram, specifically, it is an unigram which value is *Thắng*. The encoding of this unigram can described in detail as follows.

- It sets the first bit of the encoding sequence to *1* to refer that encoding for a trigram does not occur in the dictionary.
- Encoding for the number of bytes of this unigram is presented in Unicode encoding. The next seven bits are the number of bytes of this unigram in Unicode encoding; in this case, it was presented by seven bytes. So, the value of these seven bits is *0000111*.
- Encoding of this unigram. According to Unicode encoding, the encoding

sequence of this unigram is:
01010100/01101000/111000011011101010101111/01101110/01100111

- The final encoding sequence of this unigram is:
10000111/01010100/01101000/111000011011101010101111/01101110/01100111

Table 3.12 shows the compression result of all trigrams and other cases. The final encoder output sequence is:
00000000000000000000000000000000/00000000000000000000000000000001/00000000000000000000000000000010/10000111_ 01010100_ 01101000_ 111000011011101010101111_ 01101110_ 01100111

b. TGV text decompression

In this part, the TGV text decompression takes the output sequence form TGV text compression and decodes it. Its input is: *00000000000000000000000000000000/00000000000000000000000000000001/00000000000000000000000000000010/10000111_ 01010100_ 01101000_ 111000011011101010101111_ 01101110_ 01100111*
The code reading unit reads the input sequence, and separates it byte to byte. The decompression unit receives this result as its input. First, it reads the first byte of the input sequence *00000000*, and the value of the first bit of this byte is zero. Therefore, it is decoded for a standard trigram. It reads the next three bytes *000000000000000000000000* and calculates the index of this trigram in the trigrams dictionary. In this case, the value of this index is zero, so the decompression unit goes to Table 3.10 to find the value according to this index. The value is *Tôi là sinh*. The decompression unit repeats these steps to decompress all trigrams and other cases. The input sequence is the concatenation of these decoded.

3.3.5 Experiments

We conduct experiments to evaluate our method, using a data set that is a randomized collection from Vietnamese text and online newspapers. The data set includes 10 files completely different in size and content.

In order to evaluate the effects of trigrams in the dictionary (the size of dictionary), we conducted two experiments with dictionary one and dictionary two according to Table 3.9. We show the results of the two experiments in Table 3.13. Compression ratio was calculated according to Equation 3.1. According the Table 3.13, we found that the compression ratio from dictionary two is higher than the compression ratio from dictionary one. If we have a dictionary of all trigrams, the compression ratio will be better.

In Table 3.13 and Table 3.14, Figure 3.5 and 3.6, we have some abbreviations and meanings as follows: OFS: original file size, CFS: compressed file size, CR: compression ratio, D1: dictionary one, D2: Dictionary two, SB: Syllable-based method.

Table 3.13: Compression ratio of the dictionary one and the dictionary two

No.	OFS Byte	CFS-D2 Byte	CR of D2	CFS-D1 Byte	CR of D1
1	1,166	185	84.13%	305	73.84%
2	2,240	359	83.97%	719	67.90%
3	6,628	1,710	74.20%	2,404	63.73%
4	12,224	2,057	83.17%	3,321	72.83%
5	22,692	3,702	83.69%	7,469	67.09%
6	49,428	7,870	84.08%	15,872	67.89%
7	96,994	17,723	81.73%	27,161	72.00%
8	156,516	27,434	82.47%	41,228	73.66%
9	269,000	49,902	81.45%	70,105	73.94%
10	489,530	92,739	81.06%	135,639	72.29%

In Figure 3.5, the compression ratio when we use the dictionary two is higher than the compression ratio of the dictionary one.

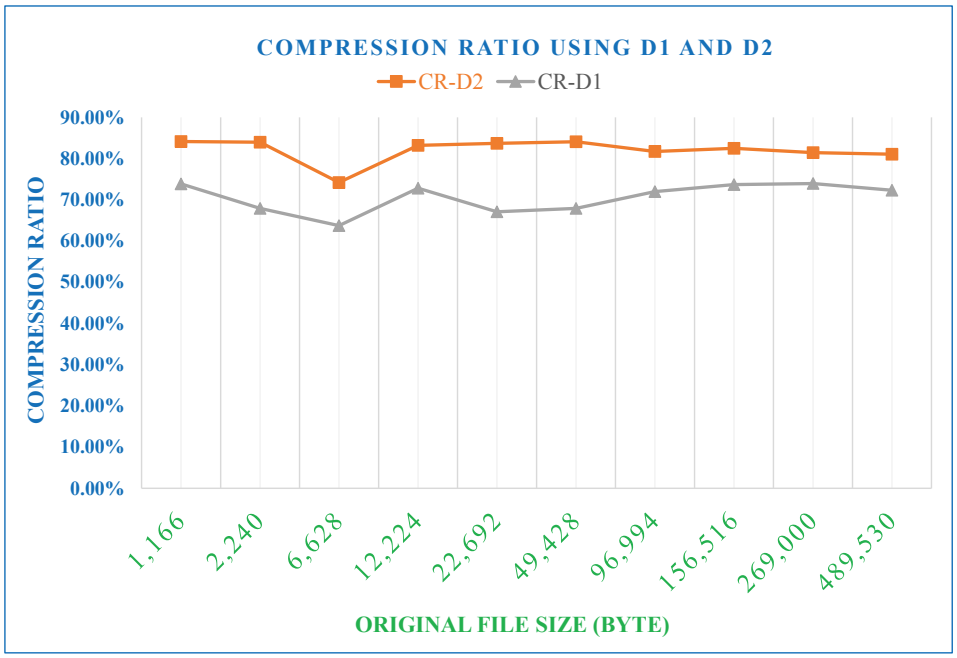

Figure 3.5: Comparison between the dictionary one and the dictionary two

Table 3.14: Compression ratio of current method and syllable-based method

No.	OFS	CFS	CR	CFS-SB	CR-SB
1	1,166	185	84.13%	345	70.41%
2	2,240	359	83.97%	599	73.26%
3	6,628	1,710	74.20%	1,803	72.80%
4	12,224	2,057	83.17%	3,495	71.41%
5	22,692	3,702	83.69%	6,418	71.72%
6	49,428	7,870	84.08%	13,881	71.92%
7	96,994	17,723	81.73%	26,772	72.40%
8	156,516	27,434	82.47%	43,701	72.08%
9	269,000	49,902	81.45%	74,504	72.30%
10	489,530	92,739	81.06%	139,985	76.25%

In order to evaluate our improvement method with the previous method, we compressed the input files using the syllable-based method in the previous section. In Table 3.14 and Figure 3.6, we show the result of our method in 10 cases above in comparison with the syllable-based method. Our improvement method achieves a higher compression ratio than the syllable-based method.

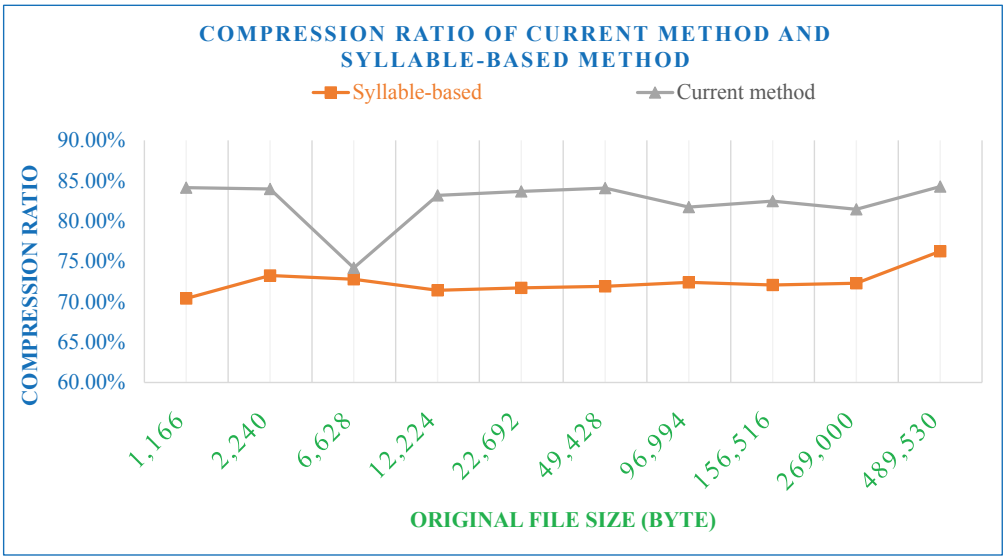

Figure 3.6: Comparison between current method and syllable-based method

3.4 N-gram based text compression

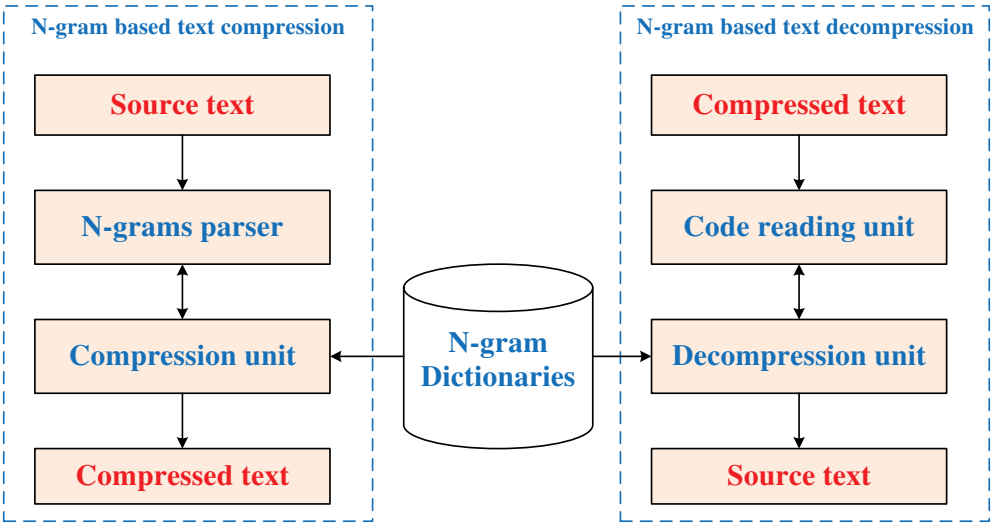

Figure 3.7: Ngram-based Vietnamese text compression model

In the previous sections we proposed Vietnamese text compression methods that are syllable-based and trigram-based. With the syllable-based method, it on the structure of the syllable and the structure of the Vietnamese morphosyllable and converges to a compression ratio around 73%. In the trigram-based method, the compression ratio is better than that of the syllable-based and normally it converges to around 82%. However, in this method, it just focuses on the trigram, and does not care about unigram, bigram, four-gram and five-gram. In this section, we present a method for Vietnamese text compression using n-gram dictionaries. This model has two main modules. The first module is used for text compression and the second module performs decompression. Figure 3.7 describes our text compression model. In our model, we use n-gram dictionaries for both compression and decompression. We will describe the model in detail in the following subsections.

3.4.1 Dictionaries

Since we focus on Vietnamese, we build five different Vietnamese dictionaries of unigram, bigram, trigram, four-gram and five-gram corresponding to the number of grams compressed. Table 1 shows these dictionaries with their number of n-grams and size. These dictionaries have been built based on a text corpus collected from the Internet. The size of the text corpus is around 2.5 GB. We use SRILM[5] to generate n-grams for these dictionaries. To increase the speed of searching in these

[5]http://www.speech.sri.com/projects/srilm/

dictionaries, we arranged them according to the alphabet. Table 3.15 describes the size and number of n-grams in each dictionary.

Table 3.15: n-grams dictionaries

n-gram dictionary	Number of n-grams	Size (MB)
1	**7,353**	**0.05**
2	**20,498,455**	**474**
3	**84,003,322**	**1,586**
4	**169,916,000**	**4,155**
5	**225,203,959**	**6,800**

Algorithm 1: Pseudo-code of the compression phase

input : The source text file
output: The encoded stream

```
1  inputstring = read source text file
2  count = number of grams in the inputstring
3  while count >= 5 do
4  |  st5 = get first five grams of the inputstring
5  |  index = find(st5, five_gram_dict)
6  |  if index>=0 then
7  |  |  force_four_gram_compression(st4)
8  |  |  outputstring += compress(index, 5)
9  |  |  delete first five grams of the inputstring
10 |  |  count -= 5
11 |  end
12 |  else
13 |  |  st4 += get first gram of the inputstring
14 |  |  delete first gram of the inputstring
15 |  |  count -= 1
16 |  |  if number of grams of st4 = 4 then
17 |  |  |  four_gram_compression(st4)
18 |  |  end
19 |  end
20 end
21 if count > 0 then
22 |  four_gram_compression(inputstring)
23 end
```

3.4.2 N-gram based text compression

As presented in Figure 3.7, the compression module takes a source text as an input, and then pass the text through two sub-modules, i.e., N-grams parser and Compression unit, to compress it. In following subsections, we explain in details.

a. N-grams parser

N-gram parser has been used to read a source text file, splits it to sentences based on newline and reads the number of grams in the combination with the result of the compression unit. In n-gram parser, we use five kinds of n-gram to store for unigram, bigram, trigram, four-gram and five-gram. Based on the result of the compression unit, the n-gram parser decides how many grams will be read next. Algorithm 1 shows the pseudo-code of this phase. If five-gram was found in the five-gram dictionary, i.e., index $>$ 0, the force_four_gram_compression function would be called to encode all previous n-grams (unigram, bigram, trigram and four-gram), then the compress function would be called to encode this five-gram. Next, the n-gram parser reads next five grams in the input string. Otherwise, it would split one leftmost gram of five-gram for four-gram and read one gram more from the input string for five-gram. When the number of grams of four-gram was 4, it calls the four_gram_compression function.

Algorithm 2: Pseudo-code of the four_gram_compression

input : The four-gram string, in this case is st4
output: The encoded stream

```
1  index = find(st4, four_gram_dict)
2  if index >= 0 then
3  |  force_trigram_compression(st3)
4  |  outputstring += compress(index, 4)
5  |  delete content of st4
6  end
7  else
8  |  st3 += first gram of st4
9  |  delete first gram of st4
10 |  if number of grams of st3 = 3 then
11 |  |  trigram_compression(st3)
12 |  end
13 end
```

Algorithm 2 shows the pseudo-code of the four_gram_compression function. This function is used to compress four-gram if it occurs in four-gram dictionary. Otherwise, it splits one leftmost gram of the four-gram variable for the trigram variable. Similar to this function, we have the trigram_compression, the bigram_compression and the unigram_compression func-

tion. The force_four_gram_compression is called to encode all four-gram, trigram, bigram and unigram when five-gram variable is found in the five-gram dictionary. Similar to this function, we have the force_trigram_compression, the force_bigram_compression and the force_unigram_compression function.

Algorithm 3: Pseudo-code of the force_four_gram_compression

input : The four-gram string, in this case is st4
output: The encoded stream

```
while number of grams of st4 > 0 do
    st3 += first gram of st4
    delete first gram of st4
    if number of grams of st3 = 3 then
        trigram_compression(st3)
    end
end
force_trigram_compression(st3)
```

b. Compression unit

The compression unit uses the result from the n-gram parser to decide how many grams will be compressed and what kind of n-gram dictionaries should be used. Based on the number of n-grams in each dictionary, we will construct the number of bytes to encode for each n-gram corresponding to the dictionary. Table 3.16 describes the number of bytes used to encode for each n-gram of each dictionary.

Table 3.16: number of encoded bytes for each n-gram of each dictionary

N-gram dictionary	Number of n-grams	Number of bytes
1	**7,353**	**2**
2	**20,498,455**	**4**
3	**84,003,322**	**4**
4	**169,916,000**	**4**
5	**225,203,959**	**4**

To classify the dictionary that was used to encode each n-gram and the other cases, we use three most significant bits (MSB) of the first byte of each encoded byte. Table 3.17 describes the value of these bits corresponding to each dictionary.

The index of each n-gram corresponding to each dictionary is encoded in the bits after the first three bits of the first byte. As seen in Table 3.17, there are two special cases for the n-gram dictionary: a newline and a unigram that doesn't

appear in the unigram dictionary corresponding to a value of "newline" and "others". In these cases, the compression unit will encode as follows:

- When the result received from the n-gram parser is the newline, the compression unit will encode the value "110" for the first three bits of MSB, and the next five bits of this byte will have the value "00000".
- When the result is the others, the three MSB of the first byte are "111" and the next five bits of this byte present the number of bytes which were used to encode this gram.

Table 3.17: value of three MSB and number of bytes

N-gram dictionary	Value of three MSB	Number of bytes is read more
1	**0 0 1**	**1**
2	**0 1 0**	**3**
3	**0 1 1**	**3**
4	**1 0 0**	**3**
5	**1 0 1**	**3**
newline	**1 1 0**	**0**
others	**1 1 1**	**value of five bits after three first bits of current byte**

3.4.3 N-gram based text decompression

As seen in Figure 3.7, the decompression module takes an compressed text as an input, and then pass the text through two sub-modules, i.e., Code reading unit and Decompression unit, to decompress it. We explain in detail in following subsections.

a. Code reading unit

First, this unit reads the compressed text from the compression phase. This result becomes the input sequence of the code reading unit. The code reading unit splits this input sequence byte to byte. Then, it reads the first byte of the input sequence, splits and analyzes the first three bits of this byte to classify the dictionary to which this n-gram belongs. Based on this result, this unit will read more bytes from the input sequence. Table 3.17 shows the number of bytes that the code reading unit reads after the first byte according to the classification of the dictionary. After reading more bytes, it transfers them to the decompression unit and repeats its work until the input sequence is null.

b. Decompression unit

This unit receives the results from the code reading unit. It decodes these results according to the classification of the dictionary as follows.

Algorithm 4: Pseudo-code of the decompression phase

```
input  : The encoded stream
output: The decoded stream
inputstring ← encodedstream
while length of inputstring > 0 do
    firstbyte = read first byte from the inputstring
    delete first byte of the the inputstring
    dict = get value of three bits of firstbyte
    if dict <= 5 then
        number = getnumberbytereadmore(dict)
        bytereadmore = read number byte more from the inputstring
        delete number byte of the inputstring
        indexstring = get last five bits of the firstbyte + the bytereadmore
        indexvalue = get value of the the indexstring
        output += decompress(indexvalue, dict)
    end
    else if dict = 6 then
        output += newline
    end
    else
        number = value of five last bits of the firstbyte
        bytereadmore = read number byte more from the inputstring
        output += decode for the bytereadmore
    end
end
```

- **Decode for n-grams occurring in dictionaries**
  - Identifying the dictionary: based on the classification dictionary from the code reading unit.
  - Identifying the index of an n-gram in the dictionary: based on the value calculated from bytes that were read by the code reading unit.
  - Decode for n-gram: when the classification of the dictionary has a value from one to five, the decompression unit decodes the n-gram in the dictionary based on the index of the n-gram.
- **Decode for n-grams that don't occur in dictionaries**

- Decode for newline: when the classification of dictionary is a "newline", it means that the value of the first three bits is 110. The decompression unit decodes a newline for this n-gram.
- Decode for others: when the classification of the dictionary is an "others", based on the value of the remaining bits of the first byte, the decompression unit will decode for all bytes after the first byte.

After finishing the decoding for one n-gram or other cases, the decompression unit reads the next result from the code reading unit and repeats the decompression tasks to decode for other n-grams or other cases until it reads the last byte. Algorithm 4 shows the pseudo-code of the decompression phase.

3.4.4 Example

a. Compression phase

Let us encode the following sequence using the n-gram approach.

Nén dữ liệu nhằm giảm kích thước dữ liệu để tăng tốc độ truyền cũng như tiết kiệm không gian lưu trữ

Assume that we have five dictionaries for unigram, bigram, trigram, four-gram and five-gram, as seen in Table 3.18.

Table 3.18: Five dictionaries

Index	entry
1	nhằm
2	lưu
3	trữ

1. uni-gram dictionary

Index	Entry
1	cũng như

2. bi-gram dictionary

Index	Entry
1	Nén dữ liệu

3. tri-gram dictionary

Index	Entry
1	tiết kiệm không gian

4. four-gram dictionary

Index	Entry
1	giảm kích thước dữ liệu
2	để tăng tốc độ truyền

5. five-gram dictionary

The n-gram parser first encounters the first five-gram *Nén dữ liệu nhằm giảm* and copies it to the five-gram variable. This pattern isn't in the five-gram dictionary, so it splits the first gram of this pattern for the four-gram variable and concatenates the next gram of the input sequence to the five-gram variable. The content of the five-gram and four-gram variables become *dữ liệu nhằm giảm kích* and *Nén*, respectively. Then, it checks the number of grams in the four-gram variable, which is one at this time. In this case, the value is less than four, it bypasses the four_gram_compression and turns back to the five-gram variable. Because this

pattern isn't in the five-gram dictionary, similar to the first case, it splits the first gram of this five-gram to the four-gram variable and concatenates the next gram of the input sequence to the five-gram variable. The content of the five-gram and four-gram variables shall become *liệu nhằm giảm kích thước* and *Nén dữ*, respectively.

Table 3.19: all steps and values of n-grams

step	five-gram variable	four-gram variable	trigram variable	bigram variable
1	Nén dữ liệu nhằm giảm			
2	dữ liệu nhằm giảm kích	Nén		
3	liệu nhằm giảm kích thước	Nén dữ		
4	nhằm giảm kích thước dữ	Nén dữ liệu		
5	giảm kích thước dữ liệu	dữ liệu nhằm	Nén	
6.1	giảm kích thước dữ liệu	liệu nhằm	Nén dữ	
6.2	giảm kích thước dữ liệu	nhằm		
6.3	giảm kích thước dữ liệu			
6.4	để tăng tốc độ truyền			
7	cũng như tiết kiệm không			
8	như tiết kiệm không gian	cũng		
9	tiết kiệm không gian lưu	cũng như		
10	kiệm không gian lưu trữ	cũng như tiết		
11	không gian lưu trữ	như tiết kiệm	cũng	
12	gian lưu trữ	tiết kiệm không	cũng như	
13.1	lưu trữ	tiết kiệm không gian		
13.2				lưu trữ
13.3				trữ
13.4				

Then, it checks the number of grams in the four-gram variable, which is two now. This value is less than four, similar to the first case, it turns back to five-gram variable. It repeats these operations until the content of the five-gram variable is *nhằm giảm kích thước dữ* and the four-gram variable is *Nén dữ liệu*. This five-gram pattern isn't in five-gram dictionary, so it splits the first gram of this

pattern for the four-gram variable and concatenates the next gram of the input sequence to the five-gram variable. The content of the five-gram and four-gram variables shall become *giảm kích thước dữ liệu* and *Nén dữ liệu nhằm*, respectively. It checks the number of grams in the four-gram variable, which is four now. It calls the four_gram_compression as presented in Algorithm 2. The four_gram_compression searches the four-gram pattern in the four-gram dictionary, which isn't found in the four-gram dictionary. It splits the first gram of this pattern into the trigram variable. The content of the four-gram and the trigram variable become *dữ liệu nhằm* and *Nén*, respectively. Then, it checks the number of grams in the trigram variable, which is one at this time. So, it bypasses the trigram_compression, exits the four_gram_compression and turns back to five-gram variable in Algorithm 1. The first five steps as seen in Table 3.19 show the content of the five-gram, four-gram, and trigram variables throughout these steps.

At Step six, first, the n-gram parser checks the value of the five-gram variable in the five-gram dictionary. This pattern is in the dictionary, therefore, it calls the compression unit to encode all bigram, trigram, and four-gram. Then, it encodes for the five-gram. When the compression unit is finished, the n-gram parser reads the next five grams from the input sequence. In Table 3.19, Steps 6.1 to 6.4 show all sub-steps of Step 6 and in Table 3.20, Steps 6.2 to 6.4 show the encoder output sequence.

Table 3.20: Encoder output sequence

step	**encoding of dictionary**	**encoded sequence**
6.2	011	00000000000000000000000000001
6.3	001	0000000000001
6.4	101	00000000000000000000000000001
7	101	00000000000000000000000000010
13.1	010	00000000000000000000000000001
13.2	100	00000000000000000000000000001
13.3	001	0000000000010
13.4	001	0000000000011

As seen in Table 3.19, at Step 6.1, the n-gram parser splits the first gram of the four-gram variable for the trigram variable, and the content of the four-gram and trigram variable shall become *liệu nhằm* and *Nén dữ*, respectively. Then, it checks the number of grams in the trigram variable, which is two at this time. So, it bypasses the trigram_compression and moves to Step 6.2. At Step 6.2, it continues splitting the first gram of the four-gram variable for the trigram variable.

The content of the four-gram and trigram variables shall become *nhằm* and *Nén dữ liệu*, respectively. Next, it checks the number of grams in the trigram variable, which is three at this time. It then searches for this trigram in the trigram dictionary. Because this trigram is in the trigram dictionary, it calls the compression unit to encode for bigram in the bigram variable. In this case, the bigram variable is null. It calls the compression unit to encode for the trigram in the trigram variable and moves to the next sub-step. The encoded sequence of this trigram is shown in Table 3.20 at Step 6.2. The first three bits of this encoded sequence which have value *011* refer to trigram dictionary as seen in Table 3.17 and all remaining bits refer to the index of this trigram in the trigram dictionary. At Step 6.3, the bigram and trigram variables are null, it counts the number of grams in the four-gram variable, which is 1 in this case, then it copies this gram to the unigram and searches for this unigram in the unigram dictionary. This unigram is in dictionary so it calls the compression unit to encode for this unigram. The encoder output sequence of this unigram is shown in Table 3.20 at Step 6.3. At Step 6.4, it calls the compression unit to encode for the five-gram in the five-gram variable, and the encoder output sequence of this five-gram is shown in Table 3.20 at Step 6.4. Then it reads the next five-gram in the input sequence to the five-gram variable. At this time, the content of the five-gram variable is *để tăng tốc độ truyền*.

The n-gram parser and the compression unit will process similar to previous cases for all remaining grams of the input sequence. The results of these steps are shown in Table 3.19 from Step 7 to Step 13.4. The encoder output sequences are shown in Table 3.20 from Step 7 to Step 13.4. The final encoder output sequence is the result of concatenation of all encoder output sequences from Step 6.1 to 13.4 in Table 3.20. The final encoder output sequence is:
01100000000000000000000000000001/00100000000000001/10100000000000000000000000001/10100000000000000000000000000010/01000000000000000000000000000001/10000000000000000000000000000001/00100000000000010/00100000000000011.

b. Decompression phase

In this section, the encoder output sequence from the previous example is taken and is decoded using the decompression unit. The encoder output sequence in the previous example was:
01100000000000000000000000000001/00100000000000001/10100000000000000000000000001/10100000000000000000000000000010/01000000000000000000000000000001/10000000000000000000000000000001/00100000000000010/00100000000000011.

The decompression unit uses the same dictionaries as the compression unit as seen in Table 3.18. It reads the first byte of the input sequence, the content of this first byte is *01100000*. The first three bits are split, and the value of these three bits is *011*. It finds the corresponding n-gram dictionary of these three bits and the number of bytes that is read more as presented in Table 3.17. In this case, the n-gram dictionary is the trigram dictionary and the number of byte that is read more is 3. The decoder reads the next three bytes from the input sequence. The

index of the entry was calculated based on the value of all remaining bits after the first three bits and the three bytes that is read more. The entry is determined based on this index. The decoder repeats these steps until it reads the last byte of the input sequence. Table 3.21 shows all steps and results of the decompression phase.

The final decoder output sequence is the result of concatenation of all decoder output sequences from Step 1 to Step 8 as presented in Table 3.21. With each decoder output sequence from Step 1 to Step 7, we add one space character before the concatenation. The final encoder output sequence is *Nén dữ liệu nhằm giảm kích thước dữ liệu để tăng tốc độ truyền cũng như tiết kiệm không gian lưu trữ*

Table 3.21: All steps and the results of the decompression phases

step	first byte	dict.	nbm	bits to calculate index	index value	decoder output sequence
1	01100000	011	3	00000000000000000000000000001	1	Nén dữ liệu
2	00100000	001	1	0000000000001	1	nhằm
3	10100000	101	3	00000000000000000000000000001	1	giảm kích thước dữ liệu
4	10100000	101	3	00000000000000000000000000010	2	để tăng tốc độ truyền
5	01000000	010	3	00000000000000000000000000001	1	cũng như
6	10000000	100	3	00000000000000000000000000001	1	tiết kiệm không gian
7	00100000	001	1	0000000000010	2	lưu
8	00100000	001	1	0000000000011	3	trữ

3.4.5 Experiments

We conducted an experiment to evaluate our method, using a data set that is randomized collection from some Vietnamese news agencies. The data set includes 10 files completely different in size and content.

In order to evaluate the effects of a combination of various n-gram dictionaries, we conducted three experiments with three kinds of systems. In the first case, we build a system with unigram, bigram, and trigram dictionaries. Next, we extend the first one with four-gram dictionary. Lastly, we extend the second one with five-gram dictionary. The results of the three experiments is shown in Table 3.22. As presented in Table 3.22, we find out that the compression ratio from the third case is the best, follow-up is the second case, and the last one comes from the first case. The compression ratio in this section was used according to Equation 3.1. In Table

3.22, 3.23, 3.24, Figure 3.8, 3.9, 3.10, we have some abbreviations and meanings as follows: OFS: original file size in byte; CFS: compressed file size in byte; CR: compression ratio; C1, C2, C3: three cases above, respectively; O: our method, S: syllable-based method, T: trigram-based method, RAR: WinRAR, ZIP: WinZIP.

Table 3.22: Compression ratio of three experience cases

No.	OFS	CFS-C1	CR-C1	CFS-C2	CR-C2	CFS-C3	CR-C3
1	1,166	210	81.99%	166	85.76%	136	88.34%
2	2,240	362	83.84%	274	87.77%	222	90.09%
3	6,628	1,245	81.22%	999	84.93%	887	86.62%
4	12,224	1,954	84.02%	1,503	87.70%	1,179	90.36%
5	22,692	3,565	84.29%	2,652	88.31%	2,180	90.39%
6	49,428	7,638	84.55%	5,712	88.44%	4,538	90.82%
7	96,994	15,636	83.88%	12,359	87.26%	10,416	89.26%
8	156,516	24,974	84.04%	19,188	87.74%	15,889	89.85%
9	269,000	43,887	83.69%	34,182	87.29%	28,937	89.24%
10	489,530	80,685	83.52%	63,472	87.03%	54,117	88.95%

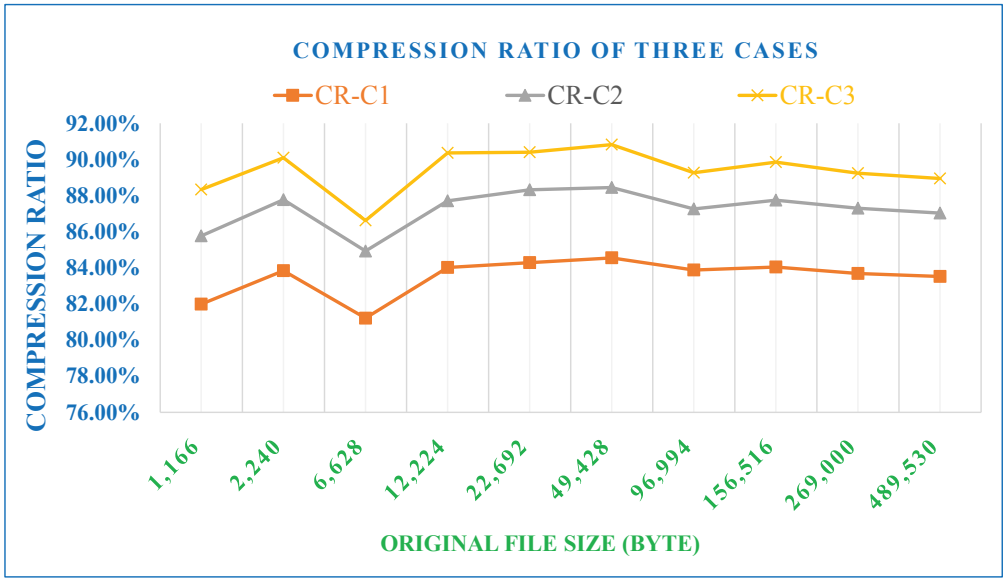

Figure 3.8: Comparison between the three cases

Table 3.23: Compression ratio of the current method with the two previous methods

No.	OFS	CFS-S	CR-S	CFS-T	CR-T	CFS-O	CR-O
1	1,166	345	70.41%	185	84.13%	136	88.34%
2	2,240	599	73.26%	359	83.97%	222	90.09%
3	6,628	1,803	72.80%	1,710	74.20%	887	86.62%
4	12,224	3,495	71.41%	2,057	83.17%	1,179	90.36%
5	22,692	6,418	71.72%	3,702	83.69%	2,180	90.39%
6	49,428	13,881	71.92%	7,870	84.08%	4,538	90.82%
7	96,994	26,772	72.40%	17,723	81.73%	10,416	89.26%
8	156,516	43,701	72.08%	27,434	82.47%	15,889	89.85%
9	269,000	74,504	72.30%	49,902	81.45%	28,937	89.24%
10	489,530	139,985	71.40%	92,739	81.06%	54,117	88.95%

In Figure 3.8, the compression ratio when we combine all five dictionaries is the highest.

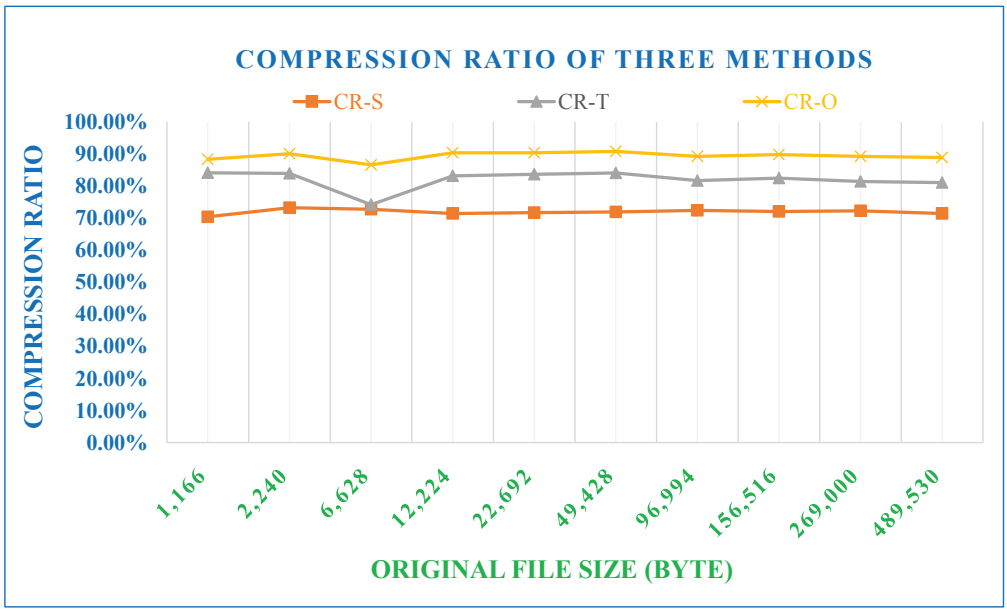

Figure 3.9: Comparison of compression ratio of three methods

In order to evaluate our method with the two previous methods, we compress the input files using these methods. In Table 3.23, we show the results of the current

method in 10 cases above in comparison to the syllable-based and trigram-based methods. As seen in Table 3.23 and Figure 3.9, the compression ratio of this method is better than the two previous methods for any size of text in our test cases.

Table 3.24: Compression ratio of our method, WinRAR and WinZIP

No.	OFS	CFS-O	CR-O	CFS-RAR	CR-RAR	CFS-ZIP	CR-ZIP
1	1,166	136	88.34%	617	47.08%	676	42.02%
2	2,240	222	90.09%	887	60.40%	946	57.77%
3	6,628	887	86.62%	2,052	69.04%	2,111	68.15%
4	12,224	1,179	90.36%	3,378	72.37%	3,442	71.84%
5	22,692	2,180	90.39%	6,162	72.85%	6,150	72.90%
6	49,428	4,538	90.82%	12,504	74.70%	12,286	75.14%
7	96,994	10,416	89.26%	21,389	77.95%	21,321	78.02%
8	156,516	15,889	89.85%	34,162	78.17%	34,362	78.05%
9	269,000	28,937	89.24%	56,152	79.13%	57,671	78.56%
10	489,530	54,117	88.95%	101,269	79.31%	108,175	77.90%

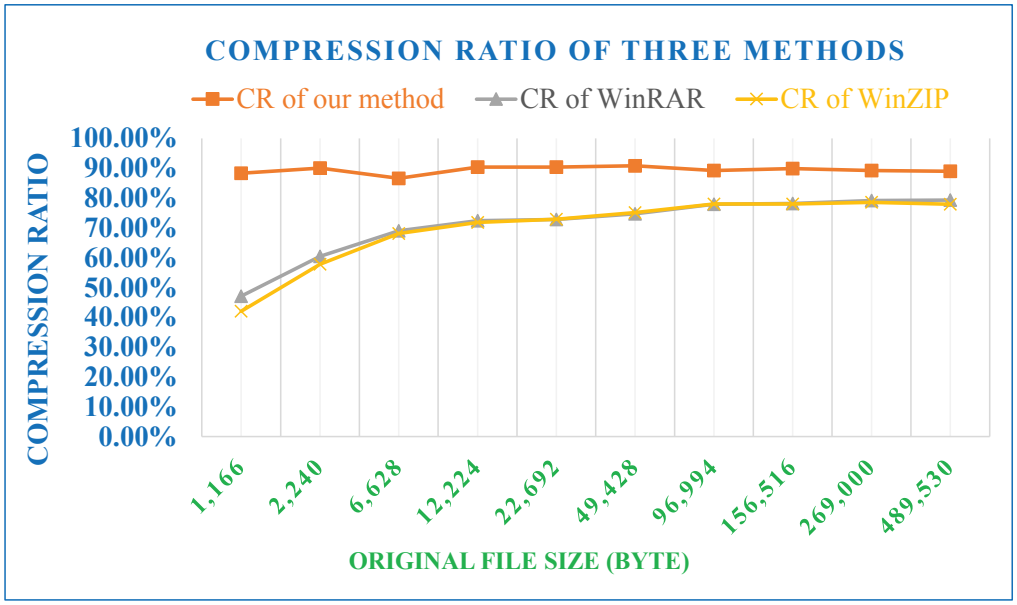

Figure 3.10: Compression ratio of our method, WinRAR and WinZIP

Table 3.24 and Figure 3.10 show the results of our method in comparison with those of other methods, such as WinZIP version 19.5[6], the software combines LZ77 [Ziv 1978] and Huffman coding, and WinRAR version 5.21[7], the software combines LZSS [Storer 1982] and Prediction by Partial Matching [Cleary 1984]. The experimental results show that our method achieves the highest compression ratio on the same testing set.

3.5 Summary

In this chapter, we presented some methods to Vietnamese text compression. Each method has its own strengths and weaknesses. The application of each method depends on the purpose of the users. The syllable-based method uses a small dictionary of all Vietnamese morphosyllables with more than 7,300 morphosyllables, and the compression ratio of this method converges to 73%. This method is useful when applied to small files, personal systems, or systems that focus more on the speed of compression. The trigram- and n-gram-based methods are suitable for systems that need a high compression ratio, however.

[6] http://www.winzip.com/win/en/index.htm
[7] http://www.rarlab.com/download.htm

Chapter 4

Normalization of Vietnamese informal text

Contents

4.1 Introduction

As mentioned in 1.1, the Vietnamese informal texts contain many spelling errors, creating a significant challenge for NER. In this chapter, we propose a method to normalize the Vietnamese informal text focusing on Vietnamese tweets by detecting non-standard words as well as spelling errors and correcting them. The method helps improve the performance of NER as well as other informal text analysis applications. There have been many methods proposed for text normalization. Most of them are to normalize texts written in English [Banko 2001, Carlson 2007, Duan 2012, Golding 1999, Habash 2011, Pennell 2014, Sonmez 2014, Sproat 2001, Yang 2013] and some other languages such as Chinese [Huang 2014, Wu 2010, Yeh 2013] or Arabic [Hassan 2014, Shaalan 2012]. We also found several methods proposed for Vietnamese spell checking in literature. However, those methods did not take informal text postings, especially in online social networks, into account.

In this chapter, we propose a system to normalize Vietnamese informal text. It consists of three steps: 1) preprocessing Vietnamese tweets; 2) detect non-standard words and misspellings due to typos; and 3) normalize and correct those errors. Table 4.1 shows an example of normalization of Vietnamese tweets according to these three steps. In Table 4.1, VNT means Vietnamese tweets, NSW means non-standard words, and MSW means misspelled words.

Table 4.1: An example of Vietnamese tweets normalization

Original tweet	**Step 1: VNT preprocessing**	**Step 2: Detect NSW & MSW**	**Step 3: Normalize**
Trời ddang mưa http://t.co/SfMgc	Trời ddang mưa	Trời ddang mưa	Trời đang mưa It is raining
ngày maii là Têt rồi http://t.co/bvgc1	ngày maii là Têt rồi	ngày maii là Têt rồi	ngày mai là Tết rồi tomorrow is our Tet's holiday
Anh yêuuu emm-mmm	Anh yêu em	Anh yêu em	anh yêu em I love you

In this chapter, we also propose a method to improve the similarity coefficient of two morphosyllables according to the Dice coefficient [Dice 1945]. When applying our improved method, the similarity coefficient increases significantly.

The rest of this chapter is organized as follows. In section 4.2 we briefly review the normalization methods. Then, we give a detailed description of our proposed method in Section 4.3. In Section 4.4, we demonstrate the experimental results for the proposed method based on the analysis of performance of our method with several other methods. Finally, we present my summaries in Section 4.5.

4.2 Related Work

Currently, there are many studies on spelling error detection and normalization. Normally, each study focuses on a specific language. For example, in the domain of English, most earlier work on automatic error correction addressed spelling errors and built models of correct usage based on native English data ([Golding 1999] [Carlson 2007] [Banko 2001]). In [Sproat 2001], to normalize nonstandard words, they developed a taxonomy of non-standard words and then they investigated the application of several general techniques, including n-gram language models, decision trees, and weighted finite-state transducers to the entire range of non-standard word types. In ([Habash 2011] and [Duan 2012]), they used the discriminative model to propose a mechanism for error detection and error correction at word-level, respectively. In [Sonmez 2014], they proposed a graph-based text normalization method that utilizes both contextual and grammatical features of text. A log-linear model was used in [Yang 2013] to characterize the relationship between standard and non-standard tokens. In [Pennell 2014], they proposed a two character-level method for the abbreviation modeling aspect of the noisy channel model, including a statistical classifier using language-based features to decide whether a character is likely to be removed from a word, and a character-level machine translation model. With Chinese language, the majority of studies used model language processing

([Wu 2010] [Yeh 2013] [Huang 2014]) and [Chen 2014] used an unsupervised model and discriminative re-ranking. With the Arabic language, recent studies used supervised learning [Hassan 2014], a character-based language model [Shaalan 2012]. Considering the Vietnamese language, we had several studies involved analyzing the word, phrase, sentence analysis, handling ambiguity, construction dictionary (VLSP[2] [Hai 1999] [Duy 2004]) and most recently, studies using the n-gram language model ([Quang 2012] [Huong 2015]).

In the field of informal texts and social networks, several researchers have focused on languages other than Vietnamese. For example, [Bo 2011, Han 2013] proposed a method to detect and handle errors based on the morphophonemic similarity. [Choi 2014] detected and handled many non-standard words in online social networks by using a diverse coefficient method, such as Dice, Jaccard, and Ochiai. [Hassan 2013] used random walks on a contextual similarity bipartite graph constructed from n-gram sequences on large unlabeled text corpus to normalize social text. [Sproat 2001] developed a novel method for normalizing and morphologically analyzing Japanese noisy text by generating both character-level and word-level normalization candidates and using discriminative methods to formulate a cost function. An approach to normalize Twitter messages in Malay based on corpus-driven analysis was proposed in [Saloot 2014]. [Cotelo 2015] proposed a modular approach for lexical normalization that was applied to Spanish tweets, in which the system proposed including the detection of modules and candidates for correction for each out-of-vocabulary word and ranking the candidates to select the best one. [Liu 2012a] proposed a normalization system for short message service (SMS) and Twitter data based on a broad-coverage normalization system by integrating three human perspectives, i.e., enhanced letter transformation, visual priming, and string/phonetic similarity.

Recently, in the Shared Tasks of the 2015 Workshop on Noisy User-generated Text: Twitter Lexical Normalization and Named Entity Recognition, several methods were proposed for the normalization of Twitter lexical usages. According to the summary of results in [Baldwin 2015], the common approaches were lexicon-based methods, CRF, and neural network-based methods. Among the constrained systems, neural networks achieved strong results. In contrast, CRF and lexicon-based approaches were shown to be effective in the unconstrained category. Considering the Vietnamese language specifically, we have not found any research work that has been undertaken for informal text.

4.3 Normalization of Vietnamese informal text

In this section, we present our method for normalization of Vietnamese informal text, focused on Vietnamese tweets. This method has three main parts, preprocessing Vietnamese tweets, detecting spelling errors, and error correction. Figure 4.1 describes our model. We describe more detail in the following subsections.

[2]http://vlsp.vietlp.org:8080/demo/

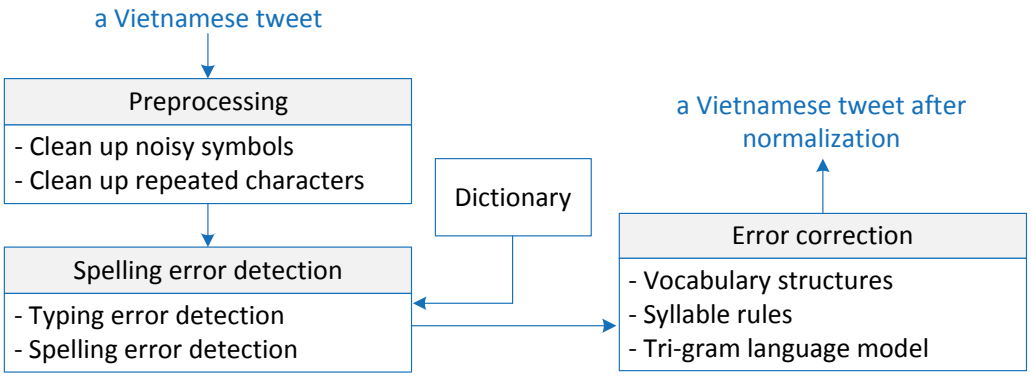

Figure 4.1: Normalization model for Vietnamese tweets

4.3.1 Preprocessing

a. Clean up noisy symbols

The original tweet can contain various noisy content, such as emotion symbols, e.g., ❤❤; the hashtag symbol, e.g., #news,#movie; link urls, e.g., https://t.co/b02RI2yXrp, https://t.co/7qod8hyIZC; etc. Those noisy symbols can affect the accuracy of the system. Therefore, in our system, first, we clean up those noisy symbols. For example, we have the tweet *Clinton hits Sanders on gun control, sharpens attacks https://t.co/b02RI2yXrp*. After clean-up of noisy symbols, it will become *Clinton hits Sanders on gun control, sharpens attacks*.

b. Clean up repeated characters

Many tweets have repeated characters, e.g., *Anh yêuuuuuuuu emmmm nhiềuuuuuuuuuu lắmmmmmmmmmm*, to express the user's feelings. These repeated characters do not have meaning and affect the accuracy of the system. Therefore, we also need to clean up these tweets. For example, with the previous tweet, after cleaning, it will become *Anh yêu em nhiều lắm* (I love you so much). We can clean those tweets based on features and properties of vowels and consonants in the Vietnamese language. Normally, Vietnamese vowels do not appear more than once except for the vowel "o", which occurs two times in the syllables "ooc" and "oong". Vietnamese consonants just appear once.

4.3.2 Spelling errors detection

Spelling errors are a big problem for any information extraction system and NER is not an exception. There are several methods to detect spelling errors. In our method, a morphosyllable in a tweet will be identified as an error if it does not appear in the standard morphosyllables dictionary in section 2.1.3. After a morphosyllable in a tweet is identified as an error, it will be analyzed to classify the

error and process it. Normally, Vietnamese morphosyllables in tweets include two kind of errors, i.e., typing errors and spelling errors. We will describe in detail the two kinds of errors in the following subsections.

a. Typing error

As we mentioned in section 2.1.2, Vietnamese uses Telex typing and VNI typing to compose Vietnamese tweets. Each method has its own combination to forming syllables and their marks. Tweets are very short and are prepared quickly, so typing speed can cause errors. For example:

- With the morphosyllable, "Nguyễn," we could have typing errors such as "nguyeenx," "nguyênx," or "nguyeenxx" with Telex typing, and "nguye6n4," "nguyên4," or "nguye6n44" with VNI typing.
- With the morphosyllable, "người", we could type the followings: "ngươif," "ngươfi," "nguowfi," "nguowif," "nguofwi," "nguofiw," "nguoifw," "nguoiwf," or "nguowff" with Telex typing, and "nguwowi2," "ngươ2i," "nguo72i," "nguo7i2," "nguo27i," "nguo2i7," "nguoi27," or "nguoi72" with VNI typing.

To handle this issue, we built a set of syllable rules with their marks and a set of rules to map these syllables to their errors, as shown in the following examples:

- "án": "asn," "ans," "a1n," or "an1"
- "àn": "afn," "anf," "a2n," or "an2"
- "ản": "arn," "anr," "a3n," or "an3"
- "ãn": "axn," "anx," "a4n," or "an4"
- "ạn": "ajn," "anj," "a5n," or "an5"

b. Spelling errors

Spelling errors occur frequently in Vietnamese tweets. Normally, they occur due to mistakes in pronunciation. Some examples of spelling errors follow:

- Error due to using the wrong mark: "quyển sách" (book) to "quyễn sách"
- Initial consonant error: "bóng chuyền" (volleyball) to "bóng truyền"
- End consonant errror: "bài hát" (song) to "bài hác"
- Region error: "tìm kiếm" (find) to "tìm kím"

4.3.3 Error correction

For the detected typing and spelling errors, first, the system uses vocabulary structures and the set of syllable rules to normalize them. Then the system uses tri-grams dictionary to normalize these results based on the degree of similarity between them.

a. Similarity of two morphosyllables

To measure the similarity of two morphosyllables, we used the results in the research of Dice [Dice 1945] with some improvements we made. To use Dice's research, we split all the characters of the morphosyllable to bigrams. Assuming that we have two morphosyllables, i.e., "nguyen" and "nguye," the bigrams of these morphosyllables can be represented as follows: $\text{bigram}_{\text{nguyn}}$={ng, gu, uy, yn}, and $\text{bigram}_{\text{nguyen}}$={ng, gu, uy,ye,en}.

Dice Coefficient:

The Dice coefficient, developed by Lee Raymond Dice [Dice 1945], is a statistical approach for comparing the similarity of two samples. The Dice coefficient of the two morphosyllables, w_i and w_j, according to bigram can be calculated using equation 1:

$$Dice(w_i, w_j) = \frac{2\times \mid bigram_{wi} \bigcap bigram_{wj} \mid}{\mid bigram_{wi} \mid + \mid bigram_{wj} \mid} \tag{4.1}$$

where:

- $\mid bigram_{wi} \mid$ and $\mid bigram_{wj} \mid$ are the total bigrams of w_i and w_j
- $\mid bigram_{wi} \mid \bigcap \mid bigram_{wj} \mid$ are the number of bigrams which appear in w_i and w_j at the same time.

If two morphosyllables are the same, the Dice coefficient is 1. The higher of the Dice coefficient, the higher the degree of similarity and vice versa.

Proposed method to improve the Dice Coefficient:

As observed from the experimental data using the Dice coefficient, we found that, the above method will be accurate with misspelled morphosyllables that have the misspelled character at the end. When misspelled characters occur close to the last character, at least we will lose the similarity of the last two grams. For a morphosyllable that has three characters, the degree of similarity is 0. For example: Dice("rất", "rát") = 0; Dice("gân", "gần") = 0;

From the above problem, we proposed a method to improve the Dice coefficient. The improvement of coefficient was performed by combining the first character with the last character of the two morphosyllables to form a new pair of bigrams. If the two members of this pair are different, the system will use the coefficients as shown in equation (1). In contrast, we use equation (2) as below:

$$iDice(w_i, w_j) = \frac{2 \times (\mid bigram_{wi} \bigcap bigram_{wj} \mid + 1)}{\mid bigram_{wi} \mid + \mid bigram_{wj} \mid + 2} \tag{4.2}$$

Let fbigram_w be an additional bigram of w. Each fbigram is the pair of the first and the last character of w. We can express the formula for improving the Dice

coefficient as equation (3):

$$fDice(w_i, w_j) = \begin{cases} Dice(w_i, w_j) : \text{if fbigram}_{\text{wi}} \text{ is different from fbigram}_{\text{wj}} \\ iDice(w_i, w_j) : \text{Otherwise} \end{cases} \quad (4.3)$$

To illustrate the improvement of the Dice coefficient, we assumed that we have two morphosyllables to measure the degree of similarity, i.e., "nguyen" and "nguyn," as presented in the previous section, thus we have $\mid bigram_{wi} \bigcap bigram_{wj} \mid = 3$. Combining the first and the last characters of the two morphosyllables we have the new pair of bigram, which has the same result, i.e., "nn." So, using the improvement of the Dice coefficient, we have fDice("nguyen," "nguyn") = 0.727. If we use the normal coefficient of Dice we have Dice("nguyen," "nguyn") = 0.667.

Table 4.2 shows the results of measuring the similarity of two morphosyllables with the Dice coefficient and the improved Dice coefficient methods. With the improved method, the similarities are obviously improved.

Table 4.2: The results of measuring the similarity of two morphosyllables with the Dice coefficient and the improved of Dice coefficient methods.

Error morphosyllable	**Correct morphosyllable**	**Dice**	**fDice**
rat	rất	0	**0.333**
rat	rác	0	0
Nguễn	Nguyễn	0.667	**0.727**
Nguễn	Nguy	0.571	0.571
Tượg	Tượng	0.571	**0.667**
Tượg	Tương	0.286	**0.444**

b. Similarity of two sentences

Assume that we need to measure the similarity of two sentences, i.e., $S_1 = w_1, w_2, w_3, \cdots, w_n$ and $S_2 = w'_1, w'_2, w'_3, \cdots, w'_n$. We compare the similarity of each pair of morphosyllables according to the improved Dice coefficient. Then, we compute the similarity of the two sentences by Equation 4.4:

$$Sim\,(S_1, S_2) = \frac{\Sigma_{i=1}^{n} fDice\,(w_i, w'_i)}{n} \quad (4.4)$$

where:

- w_i and w'_i are the corresponding morphosyllables of S_1 and S_2.
- n is the number of morphosyllables

If two sentences are the same, their degree of similarity (Sim) is 1. The higher the Sim coefficent, the higher the degree of similarity becomes, and vice versa. Table 4.3 shows the results of the normalization of Vietnamese tweets that have spelling errors.

Table 4.3: tweets with spelling errors and their normalization.

Spelling error tweets	**Normalized tweets**
xe đón hồ ngọc hà gây tai nạn **kinhh** *hoàng: sẽ khởi tố tài xế http://fb.me/2MwvznBbj*	xe đón hồ ngọc hà gây tai nạn **kinh** hoàng: sẽ khởi tố tài xế (the car picked up ho ngoc ha caused a terrible accident: the driver will be prosecuted)
hôm nay, ***siinh*** *viên* ***ddaijj*** *học tôn* ***dduwcss*** *thắng được nghỉ học*	hôm nay, **sinh** viên **đại** học tôn **đức** thắng được nghỉ học (today, students of ton duc thang university were allowed to absent)

4.4 Experiments and results

To evaluate our method, we used a data set which randomizes collected Vietnamese tweets. The data set includes 1,360 tweets that are completely different from each other.

In order to make comparisons of the impact of the data set in the language model, we ran the test two times with the language model built from two input data sets: The first set includes 130 MB randomized data from 1,045 MB of data set mention above and the second set includes entire 1,045 MB data. The trigram model with a frequency of more than 5 times the first set is about 8 MB. In this case, we use the improved Dice coefficient to measure the similarity of the two sentences. In this test, we use the precision metric to evaluate our method.

- Precision (P): number of correctly fixed errors divided by the total number of errors detected.

The results of this test was shown in Table 4.4. From Table 4.4, the results of the trigram model with data from the second set achieved a higher accuracy than the results of the trigram model with data from the first set.

Table 4.4: The results using fDice with two data sets in the trigram model

Data set	Total error	Detected error	Correct fixed	Wrong fixed	Precision
1	1,360	1,342	1,072	270	**79.88%**
2	1,360	1,342	1,207	135	**89.94%**

To evaluate the improvement Dice coefficient with normal Dice coefficient. We ran the test with trigram model built from entire data set, i.e., the data set of 1,045 MB, using Dice and fDice to measure the similarity of two sentences. In this test, we use two more metrics Recall and Balance F-Measure with the precision metric mention above to evaluate our method.

- Recall (R): number of correctly fixed errors divided by the total error.
- Balance F-measure (F1): $F1 = \frac{2*P*R}{p+R}$

Table 4.5 shows the results of this test. The table shows that the combination of our improved Dice coefficient and the tri-gram model achieved better performance than the normal Dice coefficient with the tri-gram model.

Table 4.5: The results use fDice and Dice with tri-gram language model

Method	Precision	Recall	F-Measure
Dice	**84.8%**	**83.68%**	**84.23%**
fDice	**89.94%**	**88.75%**	**89.34%**

4.5 Summary

In this chapter, we presented the first attempt to normalize Vietnamese informal text focused on tweets on Twitter. Our proposed method combines a language model with dictionaries and Vietnamese vocabulary structures. We also extended the original Dice coefficient to improve performance of the similarity measure between two morphosyllables. To evaluate the proposed method, we built a dataset including 1,360 Vietnamese tweets. The experiment results show that our proposed method achieves relative high performance with precision approximating to 90%, recall over 88.7%, and FMeasure over 89%. Moreover, our improvement on measuring the similarity of the two morphosyllables based on the Dice coefficient outperforms the original Dice coefficient.

Chapter 5

Named entity recognition in Vietnamese informal text

Contents

5.1 Context

In recent years, social networks have become very popular. It is easy for users to share their data using online social networks. Currently, Twitter is one of the most popular social networks. According to statistics from 2011, the number of tweets was up to 140 million per day[1]. With such a huge number of tweets being posted every day, effective extraction and processing of those data will be very beneficial, especially to information extraction applications.

Twitter provides an interactive environment that allows users to create their own content through tweets. Since each tweet consists of only 140 characters, users tend to use acronyms, non-standard words, and social tokens. Therefore, the tweets contain many spelling errors, and this creates a significant challenge for NER. I recommended a method to deal with this issue in chapter 4. Several recognition methods for named entities have been proposed for tweets in English and other languages [Liu 2011, Ritter 2011, Li 2015b, Bandyopadhyay 2014, Jung 2012]. Although there have been many approaches proposed for NER in the Vietnamese formal text, none is available for Vietnamese informal text. Thus, in this chapter, we propose a method for NER in informal text focused on Vietnamese tweets to

[1] https://blog.twitter.com/2011/numbers

fill the gap. The system consists of three steps, i.e., 1) normalization of the tweets using the proposed method in chapter 4; 2) the use of a capitalization classifier; and 3) recognition of named entities. Table 5.1 shows an example of NER according to these three steps.

Table 5.1: An example of named entity recognition

Original tweet	xe đón hồ ngọc hà gây tai nạn **kinhh** hoàng: sẽ khởi tố tài xế http://fb.me/2MwvznBbj
Step 1: Normalization	xe đón hồ ngọc hà gây tai nạn **kinh** hoàng: sẽ khởi tố tài xế
Step 2: Capitalization	**X**e đón **H**ồ **N**gọc **H**à gây tai nạn kinh hoàng: sẽ khởi tố tài xế
Step 3: NEs recognition	Xe đón <**PER**>Hồ Ngọc Hà</**PER**> gây tai nạn kinh hoàng: sẽ khởi tố tài xế

In Table 5.1, step 1, the original tweet will be normalized based on the proposed method in chapter 4. In this step, first it will remove the link url, then it will identify the spelling error morphosyllables. In this case, this tweet has one spelling error morphosyllabe *"kinhh."* Next, the method uses syllable rules to correct for this error. In step 2, the tweets after normalization will be put to the capitalization classifier, which was presented in 5.2.2, to identify and capitalize suitable characters. The result of this step will be put to the NER system to recognize named entities in this tweet.

In this chapter, we present the first attempt to provide NER capability in Vietnamese informal text focused on Vietnamese tweets. The rest of this chapter is organized as follows. Section 5.2 presents earlier work related to this effort. Our proposed method is presented in Section 5.3, and the experiments and their results are provided in Section 5.4. Our summaries are presented in Section 5.5.

5.2 Proposed method

In this section, we present our method for NER in Vietnamese tweets. This model has two main parts, i.e., one for training and one for recognizing. Figure 5.1 describes our model. In our model, the gazetteers are used for both training and recognizing. We will provide more detail in the following subsections.

5.2.1 Normalization

As presented in chapter 4, Vietnamese tweets on Twitter are noisy, irregular, brief and consist of acronyms and spelling errors. Therefore, we must normalize them before using NER.

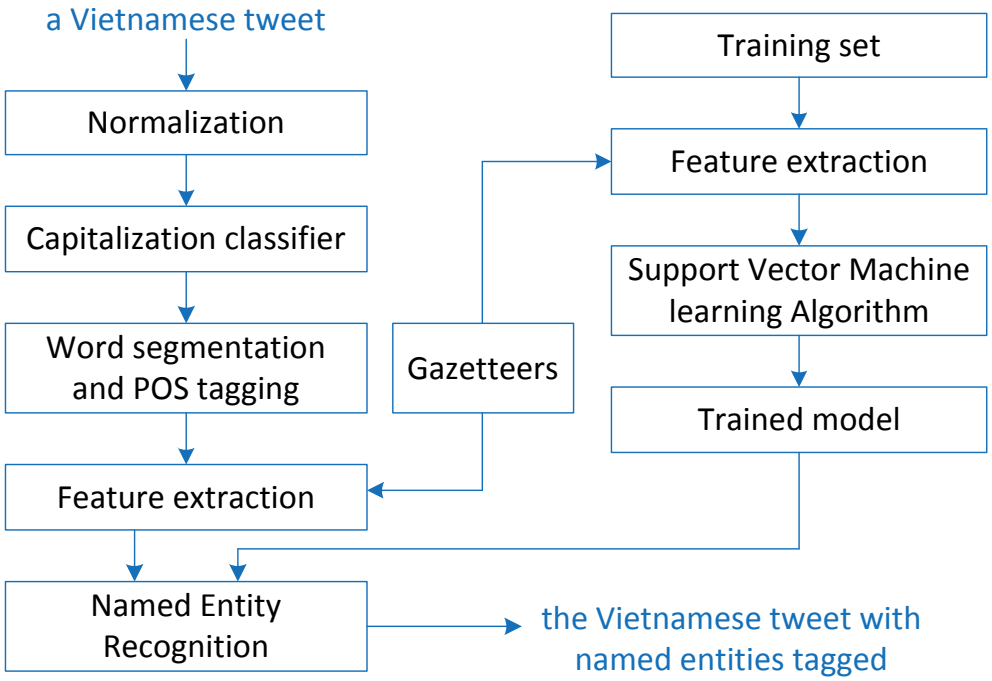

Figure 5.1: NER model in Vietnamese tweets

5.2.2 Capitalization classifier

Capitalization is a key orthographic feature for recognizing named entities [Florian 2002, Downey 2007]. Unfortunately, in tweets, capitalization is much less reliable than in edited texts. Users usually compose and reply to messages quickly, and they do not care much about capitalization. According to [Chu 2010], a letter is capitalized in the following cases:

1. Capitalize the first letter of the first syllable of a complete sentence, after punctuation (.), question mark (?), exclamation point (!), ellipsis (...) and new line.

2. Capitalize the name of people, locations, and organizations.

3. Other cases of capitalization include, e.g., medal name, position name, days of the week, months of the year, holidays, names of books, and names of magazines.

Because our method focuses on three types of entities, i.e., person, organization, and location, in the capitalization classifier, we take the first and the second cases into account. For the first case, we detect the structure of the sentence and correct

incorrect capitalization. In the second case, we use gazetteers of persons, locations, and organizations. Table 5.2 shows the results of the capitalization classifier of Vietnamese tweets.

Table 5.2: Some results of capitalization classifier of Vietnamese tweets.

Tweets before capitalization	Tweets after capitalization classifier
*xe đón **h**ồ **n**gọc **h**à gây tai nạn kinh hoàng: sẽ khởi tố tài xế*	**X**e đón **H**ồ **N**gọc **H**à gây tai nạn kinh hoàng: sẽ khởi tố tài xế (the car picked up Ho Ngoc Ha caused a terrible accident: the driver will be prosecuted)
*hôm nay, sinh viên **đ**ại học **t**ôn **đ**ức **t**hắng được nghỉ học*	**H**ôm nay, sinh viên **Đ**ại học **T**ôn **Đ**ức **T**hắng được nghỉ học (today, students of Ton Duc Thang university were allowed to absent)

5.2.3 Word segmentation and part of speech (POS) tagging

a. Word segmentation

Vietnamese is different from English and other languages in word segmentation. In English and other languages, words can be separated based on the space character. However, Vietnamese is not like that. As presented in 2.1.1, a Vietnamese word is composed of special linguistic units called Vietnamese morphosyllable. Normally, a word has from one to four morphosyllables. For example, we consider the sentence *Sinh viên Trường Đại học Tôn Đức Thắng*. In this sentence, we can separate it to these following cases:

1. *Sinh_viên Trường Đại_học Tôn_Đức_Thắng*: in this case, we separate the sentence to four words.
2. *Sinh_viên Trường_Đại_học Tôn_Đức_Thắng*: in this case it has three words.
3. *Sinh_viên Trường Đại_học_Tôn_Đức_Thắng*: in this case, it also has three words but it is different from the previous case.

The first requirement for any NER system is word segmentation. Therefore, this system is not an exception. The quality of word segmentation has an important role to the result of NER system. In our method, we used vnTokenizer[2] of [Le 2008] for word segmentation.

[2]http://mim.hus.vnu.edu.vn/phuonglh/softwares/vnTokenizer

b. POS tagging

After performing word segmentation, we apply POS tagging to give more information to the next phase. POS tagging was used to identify the characteristics of the word to enhance the accuracy of the NER system. In this stage, a word will be assigned a label according to [Le-Hong 2010], such as *Np* for proper noun, *N* for common noun, *V* for verb, etc.

For example, we apply POS tagging for the sentence result of word segmentation above: *Sinh_ viên Trường Đại_ học_ Tôn_ Đức_ Thắng.* The result of this tagging is *Sinh_ viên/N Trường/N Đại_ học_ Tôn_ Đức_ Thắng/Np*

In order for POS tagging to normalize tweets after word segmentation, we used VnTagger[3] of [Le-Hong 2010] for POS tagging.

5.2.4 Extraction of features

The aim of this phase is to convert each word to a vector of feature values. Our system uses the IOB model to annotate and assign label to data in the training and classification phases. IOB is expressed as follows:

- I: current morphosyllable is inside of a named entity (NE).
- O: current morphosyllable is outside of a NE.
- B: current morphosyllable is the beginning of a NE

Table 5.3: The characteristic value of labels according to the IOB model.

Label	Value	Meaning
O	1	Outside a named entity
B-PER	2	Beginning morphosyllable of a NE belongs to a Person class
I-PER	3	Inside morphosyllable of a NE belongs to Person class
B-LOC	4	Beginning morphosyllable of a NE belongs to Location class
I-LOC	5	Inside morphosyllable of a NE belongs to Location class
B-ORG	6	Beginning morphosyllable of a NE belongs to Organization class
I-ORG	7	Inside morphosyllable of a NE belongs to Organization class

[3] http://mim.hus.vnu.edu.vn/phuonglh/softwares/vnTagger

Table 5.3 shows the characteristic value of labels according to the IOB model with four classes, i.e., PER, LOC, ORG, and O. For example, we want to assign a label to a sentence after word segmentation from the previous section: *Sinh_viên Trường Đại_học Tôn_Đức_Thắng*. The result of this task can be described as follows.

- Trường: B-ORG
- Đại_học: I-ORG
- Tôn_Đức_Thắng: I-ORG

The selection of specific attributes from the training set has a key role in identifying the type of entity. Since the nature of the Vietnamese language is different from English, we used the most appropriate and reasonable features in order to achieve optimum accuracy for the system. Our system uses the following features:

1. **Word position**: the position of word in a sentence. For example, with the sentence above, *Sinh_viên Trường Đại_học_Tôn_Đức_Thắng*, the word *Sinh_viên* has word position value 1, the word *Trường* has word position 2 and *Đại_học_Tôn_Đức_Thắng* has word position 3.
2. **POS**: POS tag of the current word as presented in Table 5.4 according to [Le-Hong 2010].

Table 5.4: POS categories

Category	Value	Description
Np	1	Proper noun
Nc	2	Classifier
Nu	3	Unit noun
N	4	Common noun
V	5	Verb
A	6	Adjective
P	7	Pronoun
R	8	Adverb
L	9	Determiner
M	10	Numeral
E	11	Preposition
C	12	Subordinating conjunction
CC	13	Coordinating conjunction
I	14	Interjection
T	15	Auxiliary, modal words
Y	16	Abbreviation
Z	17	Bound morpheme
X	18	Unknown

3. **Orthographic (ORT)**: There are several ways to write a word in Vietnamese and it depends on the writer. For example, the word *sinh_ viên* can be written as *sinh_ viên*, *Sinh_ viên*, *Sinh_ Viên* or *SINH_ VIÊN*, etc. Therefore, in this feature, we focused on several cases of orthographics, such as capitalization of the first character of first morphosyllable, capitalization of first character of each morphosyllable, capitalization of all letters, lowercase, punctuation and numbers. The detail of categories and corresponding value of this feature was presented in Table 5.5.

Table 5.5: Orthographic categories

Label	Value	Description
I_Cap	1	Capitalization of first letter
A_Cap	2	capitalization of all letters
L_Case	3	lowercase of all letters
AF_Cap	4	capitalization of first letter of each morphosyllable
Punctuation	5	Punctuation
Digit	6	Number

4. **Gazetteer**: This feature was built based on a Gazetteer which consists of several dictionaries. Each dictionary contains words in specific types of categories such as person name, organization name, location name, prefixes, etc. These dictionaries can be updated during the process of manually annotating the corpus. When considering this feature, the system looks up the current word in these dictionaries and gets a return value to identify if this word exists in these dictionaries or not. If this word exists in these dictionaries, it will decide what kind of entity it belongs to. The detail of categories and corresponding value of Gazetteer features was presented in Table 5.6

   In this system, we built several gazetteer lists, such as person, location, organization, and prefixes. These gazetteer lists consist of more than 50,000 names of people, nearly 12,000 names of locations, and more than 7,000 names of organizations.

Table 5.6: Gazetteer categories

Category	Value	Description
PER	1	The entity is name of person
ORG	2	The entity is name of organization
LOC	3	The entity is name of location
O	4	The name entity is different from above

5. **Prefix, Suffix**: the first and the second character; the last and the next to

the last character of the current word.

6. **POS Prefix, POS Suffix**: POS tags of two previous words and POS tags of two following words of the current word.

5.3 NER training set

In Figure 5.1, before performing feature extraction, we perform word segmentation, POS tagging, and assigning labels in Table 5.3 for each word in the training set. Then, the system extracts features of the words and represents each of those words as a feature vector. A support vector machine learning algorithm was used to train the model using the training set.

In particular, we assigned labels for words in the training set by using a semi-automatic program, meaning that we assigned labels to those words with a program we wrote and checked in hand. In our self-written program, we considered the noun phrase obtained after the tagging step with a list of dictionary of text files to label for those words. The text files of the dictionary contain:

- The noun prefix for people such as you, sister, uncle, and president
- The noun prefix for organizations such as company, firm, and corporation
- The noun prefix for locations such as province, city, and district
- List of dictionary for states, provinces of Vietnam, and others

Table 5.7 shows the results of assigning labels to words of two Vietnamese tweets. The total number of entities to which we assigned labels in this phase is presented in Table 5.10.

Table 5.7: The results of assigning labels to words of two Vietnamese tweets.

Tweets	**Tweets after assigning labels**
xe đón Hồ Ngọc Hà gây tai nạn kinh hoàng: sẽ khởi tố tài xế	Xe đón <**PER**> Hồ_Ngọc_Hà </**PER**> gây tai_nạn kinh_hoàng: sẽ khởi_tố tài_xế (the car picked up Ho Ngoc Ha caused a terrible accident: the driver will be prosecuted)
hôm nay, sinh viên Đại học Tôn Đức Thắng được nghỉ học	hôm_nay, sinh_viên <**ORG**> Đại_học Tôn_Đức_Thắng </**ORG**> được nghỉ_học (today, students of Ton Duc Thang university were allowed to absent)

After assigning labels for words in Vietnamese tweets, we analyzed these tweets to build feature vectors for those words. The structure of a feature vector includes:

<label> <index1>:<value1> <index2>:<value2> <index3>:<value3> and other pairs, where:

- **<label>**: value from 1 to 7 according to 7 labels (O, B-PER, I-PER, B-LOC, I-LOC, B-ORG, I-ORG).
- **<index>:<value>**: order of feature and value corresponding to feature of a word, respectively.

To understand the process of preparation data for SVM format, we consider the following example with sentence after word segmentation above *Sinh_viên Trường Đại_học Tôn_Đức_Thắng* with four basic features: word position, POS, orthographic, and Gazetteer. Assuming that, we use the IOB model in Table 5.3, POS feature in Table 5.4, orthographic feature in Table 5.5, and Gazetteer feature in Table 5.6. The result of this process was presented in Table 5.8.

Table 5.8: The result of the process of preparation data for SVM format

Word	IOB		POS		ORT		Gazetteer	
	label	value	label	value	label	value	label	value
Sinh_viên	O	1	N	4	I_Cap	1	O	4
Trường	B-ORG	6	N	4	I_Cap	1	O	4
Đại_học	I-ORG	7	N	4	I_Cap	1	O	4
Tôn_Đức_Thắng	I-ORG	7	Np	1	AF_Cap	4	PER	1

In this sentence, *Sinh_viên Trường Đại_học Tôn_Đức_Thắng*, the position of *Sinh_viên* is 1, position of *Trường* is 2, position of *Đại_học* is 3, position of *Tôn_Đức_Thắng* is 4. Assuming that the order of features is word position, POS, ORT, and Gazetteer, the feature vectors of the words in the sentence above are presented in Table 5.9.

Table 5.9: The result of feature vectors

Word	**Feature vector**
Sinh_viên	1 1:1 2:4 3:1 4:4
Trường	6 1:2 2:4 3:1 4:4
Đại_học	7 1:3 2:4 3:1 4:4
Tôn_Đức_Thắng	7 1:4 2:1 3:4 4:1

After representing words in the training set as feature vectors, we used libSVM[4] to train the model.

[4]https://www.csie.ntu.edu.tw/ cjlin/libsvm/

Table 5.10: Total number of named entities in the training set

Entity type	**Number of named entities**
PER	10,842
LOC	19,037
ORG	12,311

5.4 Experiments

We conducted experiments with a test set including 1,668 Vietnamese tweets. To evaluate the NER method and make a comparison of the impact of the normalization of the test set, we conducted two experiments, i.e., one without normalization and capitalization classifier of tweets (Case 1) and one with normalization and capitalization classifier of tweets (Case 2). Table 5.11 shows our experimental results. In this case, we also used three metrics to evaluate our method, i.e., the precision, the recall, and the Balance F-Measure.

- Precision (P): the number of correctly recognized named entities divided by the total number of named entities recognized by the NER system.
- Recall (R): the number of correctly recognized named entities divided by the total number of named entities in the test set.
- Balance F-Measure (F1): $F1 = \frac{2*P*R}{p+R}$

Table 5.11: Experimental results of case 1 and case 2

Case	# NEs in testing set	# recognized NEs	# correctly recognized NEs	# wrong recognized NEs	P	R	F1
1	2,446	1,915	1,601	314	**83.6%**	**65.45%**	**73.42%**
2	2,446	2,266	1,939	327	**85.57%**	**79.27%**	**82.3%**

According to Table 5.11, when we applied the normalization to the test set, the precision, recall and balance F-Measure of this test were higher than the case of the test set without the application of normalization.

We re-implemented the state-of-the-art method proposed in [Tran 2007] and compared its performance with the performance of our method. The results of this comparison are shown in Table 5.12.

Table 5.12: Comparison performance of our method with that of [Tran 2007]

System	**Precision**	**Recall**	**F1**
Our system	85.57%	79.27%	82.3%
System of [Tran 2007]	83.20%	76.20%	79.55%

5.5 Summary

In this chapter, we presented the first attempt to use NER in Vietnamese informal text focused on Vietnamese tweets on Twitter. We proposed a learning model based on SVM to recognize named entities using six different types of features. In our method, we also proposed a method to capitalize for suitable characters in tweets to enhance the accuracy of the system. To evaluate the method, we built a training set of more than 40,000 named entities and a testing set of 2,446 named entities. The experimental results showed that our system achieved encouraging performance, with an 82.3% F1 score.

Chapter 6

Conclusions

In the first part of this thesis, in chapter 3, we presented our achievements in text compression. Section 3.2 proposed a method for Vietnamese text compression that focused on Vietnamese morphosyllables structure, a syllable structure used to compress Vietnamese text. In section 3.3, we proposed another method for Vietnamese text compression based on trigram. The last method that we pointed out in chapter 3 was an n-gram based. This method achieves the best compression ratio when compared with the two previous methods and it can apply to any size of text file. In the next part, in chapter 5, we presented our achievements in Named Entity Recognition (NER) for Vietnamese informal text on social networks, focused on Twitter. We also presented a method to normalize Vietnamese informal text in chapter 4 in combination with the NER model to achieve higher precision. In the next section we present in detail the main achievements of this thesis.

6.1 Thesis contributions

The contributions of this thesis can be classified in three categories.

Contributions to text compression

In this field, we present the first attempt at Vietnamese text compression. We present three methods and achieved some encouraging results based on compression ratio. In these methods, the last method achieves the best result in terms of compression ratio.

1. First we proposed a method for Vietnamese text compression based on Vietnamese morphosyllable structure, Vietnamese syllables, consonants, and vowels. To identify syllables and their marks, we built six dictionaries corresponding to six types of marks. We also proposed a method to recognize marks and capital letters. Our method gives a new approach to classifying capital letters and compressing them.

2. Next, we presented a method for Vietnamese text compression based on the trigram model. This method splits input sequences into trigrams and compresses them based on a trigrams dictionary.

3. The last method we proposed for text compression is n-gram based. In this method, we used five dictionaries from uni-gramunigram to five-gram. We

used a slide windows to identify the n-gram from input sequences and compressed them corresponding to an n-gram dictionary. This method achieved a the highest compression ratio when compared with the two previous methods. It can apply to text files with of any size and is easy to configure to compress other languages. To configure, we just collect a text corpus and build five corresponding dictionaries.

Contribution to Vietnamese informal text normalization

There are several approaches proposed to normalize for Vietnamese formal text. However, none of them proposed and applied to informal text. In this research, we proposed a method to fill the gap. This method was based on structure of Vietnamese morphosyllable, Vietnamese typing methods, syllable rules and trigrams dictionary to normalize for error morphosyllables, to identify error morphosyllables, we use a Vietnamese dictionary of standard morphosyllable was proposed in 2.1.3. In this research, we also proposed a method to improve Dice coefficient in [Dice 1945]. Our proposed method achieves a better result in the term of precision when compared with the origin method.

Table 6.1: Contributions and their publications

Contributions	**Proposed method**	**Publication**
Text compression	Syllable based method	[3]
	Tri-gram based method	[4]
	N-gram based method	[6]
Vietnamese informal text normalization	Based on Vietnamese morphosyllable structure, syllable rules, tri-grams dictionary	[2]
NER for Vietnamese informal text	combination with Vietnamese informal text normalization to normalize input data first, then using SVMs model with six different types of features to identify and classify named entities	[1], [5]

Contributions to NER in Vietnamese informal text

In this field, we present the first attempt to use NER for Vietnamese informal text focused on Twitter. To enhance the precision of the results, we combine the previous results with Vietnamese informal text normalization. Our contributions in this field can be briefly described as follows.

1. Integrate the normalization of Vietnamese informal text into NER model to normalize the input data first.

2. Propose a learning model for NER in Vietnamese informal text, focused on Vietnamese tweets, based on SVM models with six different types of features.

3. Build a training set of more than 40,000 named entities and a testing set of 2,446 named entities to evaluate the NER system of Vietnamese informal text, focused on Vietnamese tweets.

Table 6.1 shows our contributions, our proposed method, and publications in this dissertation.

6.2 Perspectives

Text compression: Text compression has an important significance on saving storage space and increasing the transfer rate. In this thesis, we have proposed three methods for text compression. Most methods have an encouraging compression ratio, especially with the n-gram based method. To achieve the best compression ratio, we must collect more text to build the n-grams dictionaries. However, when the size of the n-grams dictionaries increase, this method will face a new challenge that is the time to identify an n-gram in the n-grams dictionary. Finding the n-gram will take more time. Therefore, we need to find a method to store and index n-grams dictionaries that are easy and that will quickly identify if an n-gram occurs in the n-grams dictionaries. In the compression techniques, we can combine the current technique with an adaptive dictionary such as LZW to improve the compression ratio.

Vietnamese informal text normalization: In the normalization for Vietnamese informal text tasks, the result is quite good. However, we think we can improve it by using an n-gram slide window based on the n-grams dictionaries.

NER for Vietnamese informal text: The process of identification and classification of named entities in Vietnamese informal text plays an important role in several other systems, such as question and answering systems, machine translation systems, information retrieval systems, and more. The precision of the NER task is the main factor affecting the results of these systems. As mentioned in previous sections, the results of our method depend on the training set. Therefore, we should collect the data required to increase the number of named entities in the training set as well as to expand the dictionaries so that we can increase the NER performance of our system.

6.3 Publications

In this dissertation, we proposed methods to deal with the tasks in the thesis objective and scope. The result of these methods has been published in high-quality international conferences and journals. The publications are listed as follows.

1. Vu H Nguyen, Hien T Nguyen and Vaclav Snasel. Named Entity Recognition in Vietnamese Tweets. In The 4th International Conference on Com-

putational Social Networks, ISBN 978-3-319-21785-7, pages 205–215, China, August 2015. Springer. It was indexed in Scopus and WoS.

2. Vu H Nguyen, Hien T Nguyen and Vaclav Snasel. Normalization of vietnamese tweets on twitter. In Proceedings of the Second Euro-China Conference on Intelligent Data Analysis and Applications, ISBN 978-3-319- 21205-0, pages 179–189, Czech Republic, June 2015. Springer. It was indexed in Scopus and WoS.

3. Vu H Nguyen, Hien T Nguyen, Hieu N Duong and Vaclav Snasel. A syllable-based method for Vietnamese text compression. In Proceedings of ACM International Conference on Ubiquitous Information Management and Communication, ISBN 978-1-4503-4142-4, 6 pages, Vietnam, January 2016. ACM. It was indexed in Scopus.

4. Vu H Nguyen, Hien T Nguyen, Hieu N Duong and Vaclav Snasel. Trigram-Based Vietnamese Text Compression. In 8th Asian Conference on Intelligent Information and Database Systems, ISBN 978-3-319-31276- 7, pages 297–307, Vietnam, March 2016. Springer. It was indexed in WoS.

5. Vu H Nguyen, Hien T Nguyen and Vaclav Snasel. Text Normalization for Named Entity Recognition in Vietnamese tweets. Computational Social Networks, ISSN 2197-4314, 2016. (accepted with minor revision)

6. Vu H Nguyen, Hien T Nguyen and Vaclav Snasel. N-gram-based text compression. Computational Intelligence and Neuroscience, ISSN 1687-5273, 2016. It was indexed in Scopus and WoS. (accepted with minor revision)

Bibliography

[Akman 2011] Ibrahim Akman, Hakan Bayindir, Serkan Ozleme, Zehra Akin and Sanjay Misra. *A lossless text compression technique using syllable based morphology.* Int. Arab J. Inf. Technol., vol. 8, no. 1, pages 66–74, 2011. (Cited on page 9.)

[Al-Bahadili 2008] Hussein Al-Bahadili and Shakir M Hussain. *An adaptive character wordlength algorithm for data compression.* Computers & Mathematics with Applications, vol. 55, no. 6, pages 1250–1256, 2008. (Cited on page 8.)

[Asahara 2003] Masayuki Asahara and Yuji Matsumoto. *Japanese named entity extraction with redundant morphological analysis.* In Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1, pages 8–15. Association for Computational Linguistics, 2003. (Cited on page 11.)

[Baldwin 2015] Timothy Baldwin, Marie Catherine de Marneffe, Bo Han*et al.* *Shared Tasks of the 2015 Workshop on Noisy User-generated Text: Twitter Lexical Normalization and Named Entity Recognition.* In ACL-IJCNLP 2015, pages 126–135, 2015. (Cited on pages 10, 14 and 56.)

[Bandyopadhyay 2014] Ayan Bandyopadhyay, Dwaipayan Roy, Mandar Mitra and Sanjoy Saha. *Named Entity Recognition from Tweets.* In Proceedings of the 16th LWA Workshops: KDML, IR and FGWM, Aachen, Germany, September 8-10, 2014., pages 218–225, 2014. (Cited on page 63.)

[Banko 2001] Banko, Michele and Eric Brill. *Scaling to very very large corpora for natural language disambiguation.* In Proceedings of the 39th Annual Meeting on Association for Computational Linguistics, pages 26–33, 2001. (Cited on pages 54 and 55.)

[Bikel 1997] Daniel M Bikel, Scott Miller, Richard Schwartz and Ralph Weischedel. *Nymble: a high-performance learning name-finder.* In Proceedings of the fifth conference on Applied natural language processing, pages 194–201. Association for Computational Linguistics, 1997. (Cited on page 11.)

[Bo 2011] Han Bo and Timothy Baldwin. *Lexical normalisation of short text messages: Makn sens a# twitter.* In Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1, page 368–378, 2011. (Cited on page 56.)

[Borthwick 1998] Andrew Borthwick, John Sterling, Eugene Agichtein and Ralph Grishman. *Exploiting diverse knowledge sources via maximum entropy in named entity recognition.* In Proc. of the Sixth Workshop on Very Large Corpora, volume 182, 1998. (Cited on page 11.)

[Burrows 1994] Michael Burrows and David Wheeler. *A block-sorting lossless data compression algorithm*. In DIGITAL SRC RESEARCH REPORT. Citeseer, 1994. (Cited on page 8.)

[Carlson 2007] Carlson, Andrew and Ian Fette. *Memory-based context-sensitive spelling correction at web scale*. In Proceedings of the Sixth International Conference on Machine Learning and Applications, pages 166–171, 2007. (Cited on pages 54 and 55.)

[Chen 2014] Li Chen and Yang Liu. *Improving Text Normalization via Unsupervised Model and Discriminative Reranking*. In Proceedings of the ACL 2014 Student Research Workshop, pages 86–93. Association for Computational Linguistics, 2014. (Cited on page 56.)

[Cherry 2015] Colin Cherry, Hongyu Guo and Chengbi Dai. *NRC: Infused Phrase Vectors for Named Entity Recognition in Twitter*. In ACL-IJCNLP 2015, pages 54–60, 2015. (Cited on page 14.)

[Choi 2014] Choi, Kim*et al.* *A Method for Normalizing Non-standard Words in Online Social Network Services: A Case Study on Twitter*. In Context-Aware Systems and Applications Second International Conference, ICCASA 2013, pages 359–368, 2014. (Cited on page 56.)

[Chu 2010] Mai Ngoc Chu, Vu Duc Nghieu and Hoang Trong Phien. Basis of linguistics and vietnamese. Vietnam educational publisher, 2010. (Cited on page 65.)

[Cleary 1984] Cleary, John G., and Ian H. Witten. *Data compression using adaptive coding and partial string matching*. Communications, IEEE Transactions, vol. 32, no. 4, pages 396–402, 1984. (Cited on pages 8, 28 and 53.)

[Collins 1999] Michael Collins and Yoram Singer. *Unsupervised models for named entity classification*. In Proceedings of the joint SIGDAT conference on empirical methods in natural language processing and very large corpora, pages 100–110. Citeseer, 1999. (Cited on page 11.)

[Cotelo 2015] Juan M Cotelo, Fermín L Cruz, JA Troyano and F Javier Ortega. *A modular approach for lexical normalization applied to Spanish tweets*. Expert Systems with Applications, vol. 42, no. 10, pages 4743–4754, 2015. (Cited on page 56.)

[Crammer 2003] Koby Crammer and Yoram Singer. *Ultraconservative Online Algorithms for Multiclass Problems*. Journal of Machine Learning Research, vol. 3, pages 951–991, 2003. (Cited on page 12.)

[Cunningham 1999] Hamish Cunningham, Diana Maynard and Valentin Tablan. *JAPE: a Java annotation patterns engine*. 1999. (Cited on page 11.)

[Curran 2003] James R. Curran and Stephen Clark. *Language Independent NER using a Maximum Entropy Tagger*. In Proceedings of the Seventh Conference on Natural Language Learning, CoNLL 2003, Held in cooperation with HLT-NAACL 2003, Edmonton, Canada, May 31 - June 1, 2003, pages 164–167, 2003. (Cited on page 12.)

[Dice 1945] Dice and Lee R. *Measures of the amount of ecologic association between species*. Ecology, vol. 26, no. 3, pages 297–302, 1945. (Cited on pages 4, 55, 59 and 75.)

[Downey 2007] Doug Downey, Matthew Broadhead and Oren Etzioni. *Locating Complex Named Entities in Web Text*. In IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007, pages 2733–2739, 2007. (Cited on page 65.)

[Duan 2012] Duan, Huizhong*et al. A discriminative model for query spelling correction with latent structural SVM*. In Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, pages 1511–1521, 2012. (Cited on pages 54 and 55.)

[Duy 2004] Duy, N.T.N.*et al. An approach in Vietnamese spell checking. In: Vietnamese*, 2004. (Cited on page 56.)

[Etzioni 2005] Oren Etzioni, Michael Cafarella, Doug Downey, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S Weld and Alexander Yates. *Unsupervised named-entity extraction from the web: An experimental study*. Artificial intelligence, vol. 165, no. 1, pages 91–134, 2005. (Cited on page 11.)

[Fano 1949] Robert M. Fano. *The transmission of information*. Technical report, Massachusetts Institute of Technology, Research Laboratory of Electronics, 1949. (Cited on page 8.)

[Fersini 2014] Elisabetta Fersini, Enza Messina, G. Felici and D. Roth. *Soft-constrained inference for Named Entity Recognition*. Inf. Process. Manage., vol. 50, no. 5, pages 807–819, 2014. (Cited on page 12.)

[Florian 2002] Radu Florian. *Named Entity Recognition as a House of Cards: Classifier Stacking*. In Proceedings of the 6th Conference on Natural Language Learning, CoNLL 2002, Held in cooperation with COLING 2002, Taipei, Taiwan, 2002, 2002. (Cited on page 65.)

[Godin 2015] Frederic Godin, Baptist Vandersmissen, Wesley De Neve and Rik Van de Walle. *Multimedia Lab @ ACL W-NUT NER Shared Task: Named Entity Recognition for Twitter Microposts using Distributed Word Representations*. In ACL-IJCNLP 2015, pages 146–153, 2015. (Cited on page 14.)

[Golding 1999] Golding, Andrew R. and Dan Roth. *A winnow-based approach to context-sensitive spelling correction.* Machine learning, vol. 34.1-3, pages 107–130, 1999. (Cited on pages 54 and 55.)

[Grishman 1996] Ralph Grishman and Beth Sundheim. *Message Understanding Conference-6: A Brief History.* In COLING, volume 96, pages 466–471, 1996. (Cited on page 10.)

[Habash 2011] Habash, Nizar and Ryan M. Roth. *Using deep morphology to improve automatic error detection in Arabic handwriting recognition.* In Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1, pages 875–884, 2011. (Cited on pages 54 and 55.)

[Hai 1999] Hai, N.D.*et al. Syntactic parser in Vietnamese sentences and its application in spell checking. In: Vietnamese*, 1999. (Cited on page 56.)

[Han 2013] Bo Han*et al. Lexical normalization for social media text.* ACM Transactions on Intelligent Systems and Technology, vol. 4.1, pages 621–633, 2013. (Cited on page 56.)

[Hassan 2013] Hany Hassan and Arul Menezes. *Social Text Normalization using Contextual Graph Random Walks.* In Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics, page 1577–1586. Association for Computational Linguistics, 2013. (Cited on page 56.)

[Hassan 2014] Hassan, Youssef*et al. Arabic Spelling Correction using Supervised Learning.* In Proceedings of the EMNLP 2014 Workshop on Arabic Natural Language Processing, pages 121–126. Association for Computational Linguistics, 2014. (Cited on pages 54 and 56.)

[Howard 1994] Paul G Howard and Jeffrey Scott Vitter. *Arithmetic coding for data compression.* Proceedings of the IEEE, vol. 82, no. 6, pages 857–865, 1994. (Cited on page 8.)

[Huang 2014] Huang, Qiang*et al. Chinese Spelling Check System Based on Trigram Model.* In Proceedings of The Third CIPS-SIGHAN Joint Conference on Chinese Language Processing, pages 173–178, 2014. (Cited on pages 54 and 56.)

[Huffman 1952] David A. Huffman. *A method for the construction of minimum redundancy codes.* In Proceedings of the IRE, volume 40.9, pages 1098–1101, 1952. (Cited on page 8.)

[Humphreys 1998] Kevin Humphreys, Robert Gaizauskas, Saliha Azzam, Chris Huyck, Brian Mitchell, Hamish Cunningham and Yorick Wilks. *University of Sheffield: Description of the LaSIE-II system as used for MUC-7.* In

Proceedings of the Seventh Message Understanding Conferences (MUC-7). Citeseer, 1998. (Cited on page 11.)

[Huong 2015] Nguyen Thi Xuan Huong*et al.* *Using Large N-gram for Vietnamese Spell Checking.* In Proceedings of Sixth International Conference KSE 2014, pages 617–627. Springer International Publishing, 2015. (Cited on page 56.)

[Jung 2012] Jason J. Jung. *Online named entity recognition method for microtexts in social networking services: A case study of twitter.* Expert Syst. Appl., vol. 39, no. 9, pages 8066–8070, 2012. (Cited on pages 13 and 63.)

[Kalajdzic 2015] Kenan Kalajdzic, Samaher Hussein Ali and Ahmed Patel. *Rapid lossless compression of short text messages.* Computer Standards & Interfaces, vol. 37, pages 53–59, 2015. (Cited on page 9.)

[Konkol 2015] Michal Konkol, Tomas Brychcin and Miloslav Konopík. *Latent semantics in Named Entity Recognition.* Expert Syst. Appl., vol. 42, no. 7, pages 3470–3479, 2015. (Cited on page 12.)

[Lansky 2005] Jan Lansky and Michal Zemlicka. *Text Compression: Syllables.* In Proceedings of the Dateso 2005 Annual International Workshop on DAtabases, TExts, Specifications and Objects, Desna, Czech Republic, April 13-15, pages 32–45, 2005. (Cited on page 9.)

[Le-Hong 2010] Phuong Le-Hong, Azim Roussanaly*et al.* *An empirical study of maximum entropy approach for part-of-speech tagging of Vietnamese texts.* In Traitement Automatique des Langues Naturelles-TALN 2010, 2010. (Cited on pages 67 and 68.)

[Le 2008] Hong Phuong Le, Nguyên Thi Minh Huyên, Azim Roussanaly and Hô Tuòng Vinh. *A Hybrid Approach to Word Segmentation of Vietnamese Texts.* In Language and Automata Theory and Applications, Second International Conference, LATA 2008, Tarragona, Spain, March 13-19, 2008. Revised Papers, pages 240–249, 2008. (Cited on page 66.)

[Le 2011] Hoang-Quynh Le, Mai-Vu Tran, Nhat-Nam Bui, Nguyen-Cuong Phan and Quang-Thuy Ha. *An Integrated Approach Using Conditional Random Fields for Named Entity Recognition and Person Property Extraction in Vietnamese Text.* In International Conference on Asian Language Processing, IALP 2011, Penang, Malaysia, 15-17 November, 2011, pages 115–118, 2011. (Cited on pages 12 and 13.)

[Le 2013a] Huong Thanh Le, Rathany Chan Sam, Hoan Cong Nguyen and Thuy Thanh Nguyen. *Named entity recognition in vietnamese text using label propagation.* In 2013 International Conference on Soft Computing and Pattern Recognition, SoCPaR 2013, Hanoi, Vietnam, December 15-18, 2013, pages 366–370, 2013. (Cited on page 12.)

[Le 2013b] Huong Thanh Le and Luan Van Tran. *Automatic feature selection for named entity recognition using genetic algorithm.* In 4th International Symposium on Information and Communication Technology, SoICT '13, Danang, Viet Nam - December 05 - 06, 2013, pages 81–87, 2013. (Cited on page 12.)

[Le 2015] Huong Thanh Le, Luan Van Tran, Xuan Hoai Nguyen and Thi Hien Nguyen. *Optimizing Genetic Algorithm in Feature Selection for Named Entity Recognition.* In Proceedings of the Sixth International Symposium on Information and Communication Technology, Hue City, Vietnam, December 3-4, 2015, page 5, 2015. (Cited on page 12.)

[Li 2014] Chen Li and Yang Liu. *Improving Text Normalization via Unsupervised Model and Discriminative Reranking.* In Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA, Student Research Workshop, pages 86–93, 2014. (Cited on page 13.)

[Li 2015a] Chen Li and Yang Liu. *Improving Named Entity Recognition in Tweets via Detecting Non-Standard Words.* In Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing,ACL 2015, July 26-31, 2015, Beijing, China, Volume 1: Long Papers, pages 929–938, 2015. (Cited on page 13.)

[Li 2015b] Chenliang Li, Aixin Sun, Jianshu Weng and Qi He. *Tweet Segmentation and Its Application to Named Entity Recognition.* IEEE Trans. Knowl. Data Eng., vol. 27, no. 2, pages 558–570, 2015. (Cited on page 63.)

[Liao 2009] Wenhui Liao and Sriharsha Veeramachaneni. *A Simple Semi-supervised Algorithm for Named Entity Recognition.* In Proceedings of the NAACL HLT Workshop on Semisupervised Learning for Natural Language Processing, pages 28–36, 2009. (Cited on page 12.)

[Liu 2011] Xiaohua Liu, Shaodian Zhang, Furu Wei and Ming Zhou. *Recognizing Named Entities in Tweets.* In The 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, Proceedings of the Conference, 19-24 June, 2011, Portland, Oregon, USA, pages 359–367, 2011. (Cited on pages 12, 13 and 63.)

[Liu 2012a] Fei Liu, Fuliang Weng and Xiao Jiang. *A Broad-Coverage Normalization System for Social Media Language.* In The 50th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference, July 8-14, 2012, Jeju Island, Korea - Volume 1: Long Papers, pages 1035–1044, 2012. (Cited on page 56.)

[Liu 2012b] Xiaohua Liu, Ming Zhou, Xiangyang Zhou, Zhongyang Fu and Furu Wei. *Joint Inference of Named Entity Recognition and Normalization for*

Tweets. In The 50th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference, July 8-14, 2012, Jeju Island, Korea - Volume 1: Long Papers, pages 526–535, 2012. (Cited on page 14.)

[Liu 2013a] Xiaohua Liu, Furu Wei, Shaodian Zhang and Ming Zhou. *Named entity recognition for tweets*. ACM TIST, vol. 4, no. 1, page 3, 2013. (Cited on page 14.)

[Liu 2013b] Xiaohua Liu and Ming Zhou. *Two-stage NER for tweets with clustering*. Inf. Process. Manage., vol. 49, no. 1, pages 264–273, 2013. (Cited on page 14.)

[Mayfield 2003] James Mayfield, Paul McNamee and Christine D. Piatko. *Named Entity Recognition using Hundreds of Thousands of Features*. In Proceedings of the Seventh Conference on Natural Language Learning, CoNLL 2003, Held in cooperation with HLT-NAACL 2003, Edmonton, Canada, May 31 - June 1, 2003, pages 184–187, 2003. (Cited on page 12.)

[McCallum 2003] Andrew McCallum and Wei Li. *Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons*. In Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4, pages 188–191. Association for Computational Linguistics, 2003. (Cited on pages 11 and 12.)

[Mikheev 1998] Andrei Mikheev, Claire Grover and Marc Moens. *Description of the LTG system used for MUC-7*. In Proceedings of 7th Message Understanding Conference (MUC-7), pages 1–12. Fairfax, VA, 1998. (Cited on page 11.)

[Mikheev 1999] Andrei Mikheev, Marc Moens and Claire Grover. *Named entity recognition without gazetteers*. In Proceedings of the ninth conference on European chapter of the Association for Computational Linguistics, pages 1–8. Association for Computational Linguistics, 1999. (Cited on page 11.)

[Nguyen 2007a] Truc-Vien T Nguyen and Tru H Cao. *Vn-kim ie: Automatic extraction of vietnamese named-entities on the web*. New Generation Computing, vol. 25, no. 3, pages 277–292, 2007. (Cited on page 11.)

[Nguyen 2007b] Truc-Vien T. Nguyen and Tru H. Cao. *VN-KIM IE: Automatic Extraction of Vietnamese Named-Entities on the Web*. New Generation Comput., vol. 25, no. 3, pages 277–292, 2007. (Cited on pages 12 and 13.)

[Nguyen 2010] Dat Ba Nguyen, Son Huu Hoang, Son Bao Pham and Thai Phuong Nguyen. *Named Entity Recognition for Vietnamese*. In Intelligent Information and Database Systems, Second International Conference, ACIIDS, Hue City, Vietnam, March 24-26, 2010. Proceedings, Part II, pages 205–214, 2010. (Cited on pages 12 and 13.)

[Nguyen 2012a] Dat Ba Nguyen and Son Bao Pham. *Ripple Down Rules for Vietnamese Named Entity Recognition*. In Computational Collective Intelligence. Technologies and Applications - 4th International Conference, ICCCI 2012, Ho Chi Minh City, Vietnam, November 28-30, 2012, Proceedings, Part I, pages 354–363, 2012. (Cited on page 12.)

[Nguyen 2012b] Truc-Vien T. Nguyen and Tru H. Cao. *Linguistically Motivated and Ontological Features for Vietnamese Named Entity Recognition*. In 2012 IEEE RIVF International Conference on Computing & Communication Technologies, Research, Innovation, and Vision for the Future (RIVF), Ho Chi Minh City, Vietnam, February 27 - March 1, 2012, pages 1–6, 2012. (Cited on page 12.)

[Nguyen 2012c] Truc-Vien T. Nguyen and Alessandro Moschitti. *Structural reranking models for named entity recognition*. Intelligenza Artificiale, vol. 6, no. 2, pages 177–190, 2012. (Cited on page 14.)

[Nguyen 2015a] Vu H Nguyen, Hien T Nguyen and Vaclav Snasel. *Named Entity Recognition in Vietnamese Tweets*. In The 5th International Conference on Computational Social Networks, ISBN 978-3-319-21785-7, pages 205–215, China, August 2015. Springer. (Cited on page 2.)

[Nguyen 2015b] Vu H Nguyen, Hien T Nguyen and Vaclav Snasel. *Normalization of vietnamese tweets on twitter*. In Proceedings of the Second Euro-China Conference on Intelligent Data Analysis and Applications, ISBN 978-3-319-21205-0, pages 179–189, Czech Republic, June 2015. Springer. (Cited on page 2.)

[Nguyen 2016a] Vu H Nguyen, Hien T Nguyen, Hieu N Duong and Vaclav Snasel. *A syllable-based method for Vietnamese text compression*. In Proceedings of ACM International Conference on Ubiquitous Information Management and Communication, ISBN 978-1-4503-4142-4, 6 pages, Vietnam, January 2016. ACM. (Cited on page 3.)

[Nguyen 2016b] Vu H Nguyen, Hien T Nguyen, Hieu N Duong and Vaclav Snasel. *Trigram-Based Vietnamese Text Compression*. In 8th Asian Conference on Intelligent Information and Database Systems, ISBN 978-3-319-31276-7, pages 297–307, Vietnam, March 2016. Springer. (Cited on page 3.)

[Pennell 2014] Deana L Pennell and Yang Liu. *Normalization of informal text*. Computer Speech & Language, vol. 28, no. 1, pages 256–277, 2014. (Cited on pages 54 and 55.)

[Pham 2015] Quang H. Pham, Minh-Le Nguyen, Binh T. Nguyen and Nguyen Viet Cuong. *Semi-supervised Learning for Vietnamese Named Entity Recognition using Online Conditional Random Fields*. In Proceedings of NEWS 2015 The Fifth Named Entities Workshop, pages 53–58, 2015. (Cited on page 12.)

[Phe 2011] Hoang Phe. syllable dictionary. Dictionary center, Hanoi encyclopedia Publishers, fifth édition, 2011. (Cited on page 5.)

[Platos 2008a] Jan Platos and Jiri Dvorský. *Word-Based Text Compression*. CoRR, vol. abs/0804.3680, 2008. (Cited on page 9.)

[Platoš 2008b] Jan Platoš, Václav Snášel and Eyas El-Qawasmeh. *Compression of small text files*. Advanced engineering informatics, vol. 22, no. 3, pages 410–417, 2008. (Cited on page 9.)

[Quang 2012] N.H.T. Quang. *Language model and word segmentation in Vietnamese Spell checking. In: Vietnamese*, 2012. (Cited on page 56.)

[Ramage 2009] Daniel Ramage, David Leo Wright Hall, Ramesh Nallapati and Christopher D. Manning. *Labeled LDA: A supervised topic model for credit attribution in multi-labeled corpora*. In Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing, pages 248–256, 2009. (Cited on page 13.)

[Riloff 1999] Ellen Riloff, Rosie Jones*et al. Learning dictionaries for information extraction by multi-level bootstrapping*. In AAAI/IAAI, pages 474–479, 1999. (Cited on page 11.)

[Ritter 2011] Alan Ritter, Sam Clark, Mausam and Oren Etzioni. *Named Entity Recognition in Tweets: An Experimental Study*. In Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing, EMNLP 2011, 27-31 July 2011, John McIntyre Conference Centre, Edinburgh, UK, A meeting of SIGDAT, a Special Interest Group of the ACL, pages 1524–1534, 2011. (Cited on pages 13 and 63.)

[Robinson 1967] AH Robinson and Colin Cherry. *Results of a prototype television bandwidth compression scheme*. Proceedings of the IEEE, vol. 55, no. 3, pages 356–364, 1967. (Cited on page 8.)

[Salomon 2010] David Salomon and Giovanni Motta. Data compression - the complete reference. Springer, 5 édition, 2010. (Cited on page 7.)

[Saloot 2014] Saloot, Mohammad Arshi*et al. An architecture for Malay Tweet normalization*. Information Processing & Management, vol. 50.5, pages 621–633, 2014. (Cited on page 56.)

[Sam 2011] Rathany Chan Sam, Huong Thanh Le, Thuy Thanh Nguyen and Thien Huu Nguyen. *Combining Proper Name-Coreference with Conditional Random Fields for Semi-supervised Named Entity Recognition in Vietnamese Text*. In Advances in Knowledge Discovery and Data Mining - 15th Pacific-Asia Conference, PAKDD 2011, Shenzhen, China, May 24-27, 2011, Proceedings, Part I, pages 512–524, 2011. (Cited on pages 12 and 13.)

[Shaalan 2012] Shaalan, Khaled F.*et al. Arabic Word Generation and Modelling for Spell Checking.* In Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12). European Language Resources Associations, 2012. (Cited on pages 54 and 56.)

[Shannon 1948] Claude E. Shannon. *A mathematical theory of communication.* The Bell System Technical Journal, vol. 27, no. 3, pages 379 – 423, 1948. (Cited on page 8.)

[Sonmez 2014] Cagil Sonmez and Arzucan Ozgur. *A Graph-based Approach for Contextual Text Normalization.* In EMNLP, pages 313–324. Association for Computational Linguistics, 2014. (Cited on pages 54 and 55.)

[Sproat 2001] Richard Sproat, Alan W Black, Stanley Chen, Shankar Kumar, Mari Ostendorf and Christopher Richards. *Normalization of non-standard words.* Computer Speech & Language, vol. 15, no. 3, pages 287–333, 2001. (Cited on pages 54, 55 and 56.)

[Storer 1982] Storer, James A. and Thomas G. Szymanski. *Data compression via textual substitution.* Journal of the ACM, vol. 29, no. 4, pages 928–951, 1982. (Cited on pages 28 and 53.)

[Thao 2007] Pham Thi Xuan Thao, Tran Quoc Tri, Dinh Dien and Nigel Collier. *Named entity recognition in Vietnamese using classifier voting.* ACM Trans. Asian Lang. Inf. Process., vol. 6, no. 4, 2007. (Cited on pages 5, 12 and 13.)

[Tjong Kim Sang 2002] Erik F Tjong Kim Sang. *Introduction to the CoNLL-2002 Shared Task: Language-Independent Named Entity Recognition.* In Proceedings of the 6th Conference on Natural Language Learning, CoNLL 2002, Held in cooperation with COLING 2002, Taipei, Taiwan, 2002, 2002. (Cited on pages 10 and 11.)

[Tjong Kim Sang 2003] Erik F Tjong Kim Sang and Fien De Meulder. *Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition.* In Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4, pages 142–147. Association for Computational Linguistics, 2003. (Cited on pages 10 and 11.)

[Tran 2007] Q Tri Tran, TX Thao Pham, Q Hung Ngo, Dien Dinh and Nigel Collier. *Named entity recognition in Vietnamese documents.* Progress in Informatics Journal, vol. 5, pages 14–17, 2007. (Cited on pages 5, 12, 13, 14 and 73.)

[Tran 2015] Van Cuong Tran, Dosam Hwang and Jason J. Jung. *Semi-supervised approach based on co-occurrence coefficient for named entity recognition on Twitte.* In Information and Computer Science (NICS), 2015 2nd National Foundation for Science and Technology Development Conference on. IEEE, pages 141–146, 2015. (Cited on page 13.)

[Trung 2014] Hieu Le Trung, Vu Le Anh and Kien Le Trung. *Bootstrapping and Rule-Based Model for Recognizing Vietnamese Named Entity.* In Intelligent Information and Database Systems - 6th Asian Conference, ACIIDS 2014, Bangkok, Thailand, April 7-9, 2014, Proceedings, Part II, pages 167–176, 2014. (Cited on page 12.)

[Tu 2005] Tu, Nguyen Cam*et al.* *Named entity recognition in vietnamese free-text and web documents using conditional random fields.* In The 8th Conference on Some selection problems of Information Technology and Telecommunication, 2005. (Cited on pages 12 and 13.)

[Welch 1984] Terry A. Welch. *A Technique for High-Performance Data Compression.* IEEE Computer, vol. 17, no. 6, pages 8–19, 1984. (Cited on page 8.)

[Witten 1987] Ian H Witten, Radford M Neal and John G Cleary. *Arithmetic coding for data compression.* Communications of the ACM, vol. 30, no. 6, pages 520–540, 1987. (Cited on page 8.)

[Wu 2010] Wu, Shih-Hung*et al.* *Reducing the false alarm rate of Chinese character error detection and correction.* In Proceedings of CIPS-SIGHAN Joint Conference on Chinese Language Processing (CLP 2010), pages 54–61, 2010. (Cited on pages 54 and 56.)

[Yamada 2015] Ikuya Yamada, Hideaki Takeda and Yoshiyasu Takefuji. *Enhancing Named Entity Recognition in Twitter Messages Using Entity Linking.* In ACL-IJCNLP 2015, pages 136–140, 2015. (Cited on page 14.)

[Yang 2013] Yi Yang and Jacob Eisenstein. *A Log-Linear Model for Unsupervised Text Normalization.* In Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing - EMNLP, pages 61–72. Association for Computational Linguistics, 2013. (Cited on pages 54 and 55.)

[Yeh 2013] Yeh, Jui-Feng*et al.* *Chinese Word Spelling Correction Based on N-gram Ranked Inverted Index List.* In Proceedings of the Seventh SIGHAN Workshop on Chinese Language Processing (SIGHAN-7), pages 43–48, 2013. (Cited on pages 54 and 56.)

[Zhou 2002] Guodong Zhou and Jian Su. *Named Entity Recognition using an HMM-based Chunk Tagger.* In Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, July 6-12, 2002, Philadelphia, PA, USA., pages 473–480, 2002. (Cited on page 12.)

[Zirikly 2015] Ayah Zirikly and Mona Diab. *Named Entity Recognition for Arabic Social Media.* In Proceedings of NAACL-HLT 2015, pages 176–185, 2015. (Cited on page 14.)

[Ziv 1977] Jacob Ziv and Abraham Lempel. *A universal algorithm for sequential data compression.* IEEE Transactions on Information Theory, vol. 23, no. 3, pages 337–343, 1977. (Cited on page 8.)

[Ziv 1978] Jacob Ziv and Abraham Lempel. *Compression of individual sequences via variable-rate coding.* IEEE Transactions on Information Theory, vol. 24, no. 5, pages 530–536, 1978. (Cited on pages 8, 28 and 53.)